# Betriebssysteme

**it-Akademie Bayern**
**z/OS und OS/390 Lehrgang 2009**

**Prof. Dr.-Ing. Wilhelm G. Spruth**

## Teil 12

## z/OS Unix System Services

# Unix

## Welche Betriebssysteme findet man in der Praxis ?

- **Windows, verschiedene Varianten**
- **Unix, verschiedene Varianten**
- **i-Series OS/400**
- **zSeries Betriebssysteme – z/OS, zVM, zVSE, TPF**

## Welche wesentlichen Unix Varianten existieren ?

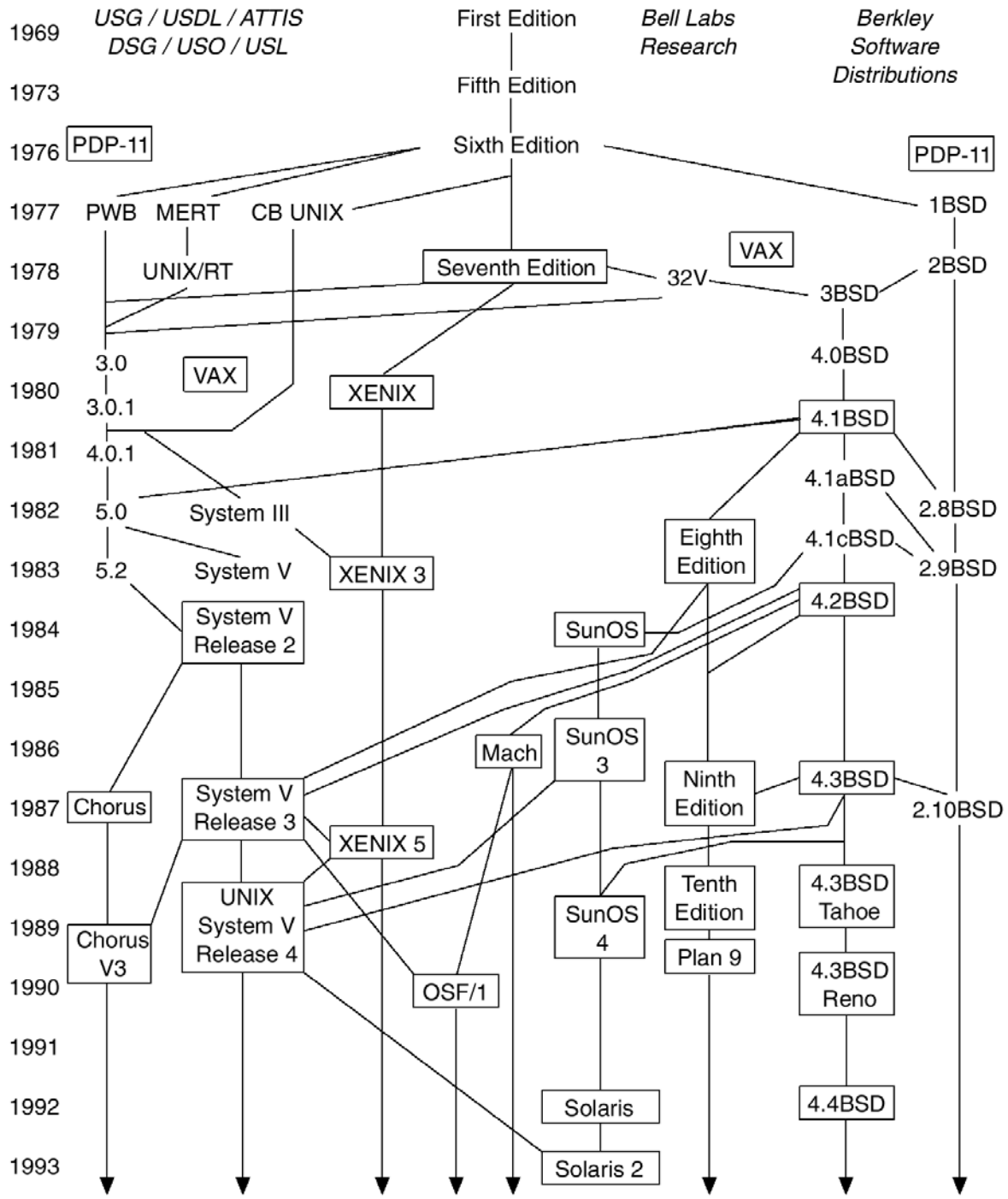
- **HP/UX**
- **SunSolaris**
- **IBM AIX**
- **Siemens Sinix**
- **MacOS (BSD)**
- **Linux, einschließlich zLinux**
- **z/OS Unix System Services**

# Literatur

**OS/390 Unix System Services. IBM Form No. SC28-1891-8**

**Jürgen Gulbins: “Unix”. Springer Verlag , 3. Auflage, 1988. ISBN 3-540-19248-4**

**Gutes Handbuch, um mit den einzelnen Unix Shell befehlen zu arbeiten.**




# History of Unix Operating Systems

# z/OS vs. Unix

**OS/390**

**mehrere 1000 E/A Operationen / s**

**Unix**

**mehrere 100 E/A Operationen / s**

# Native Unix Betriebssysteme für S/390

**Amdahl UTS (Universal Time Sharing System)**
**Marktführer, < 300 Installationen**

**Hitachi HI-OSF/1-M**

**IBM AIX/ESA (nicht mehr verfügbar)**

# z/OS Unix System Services

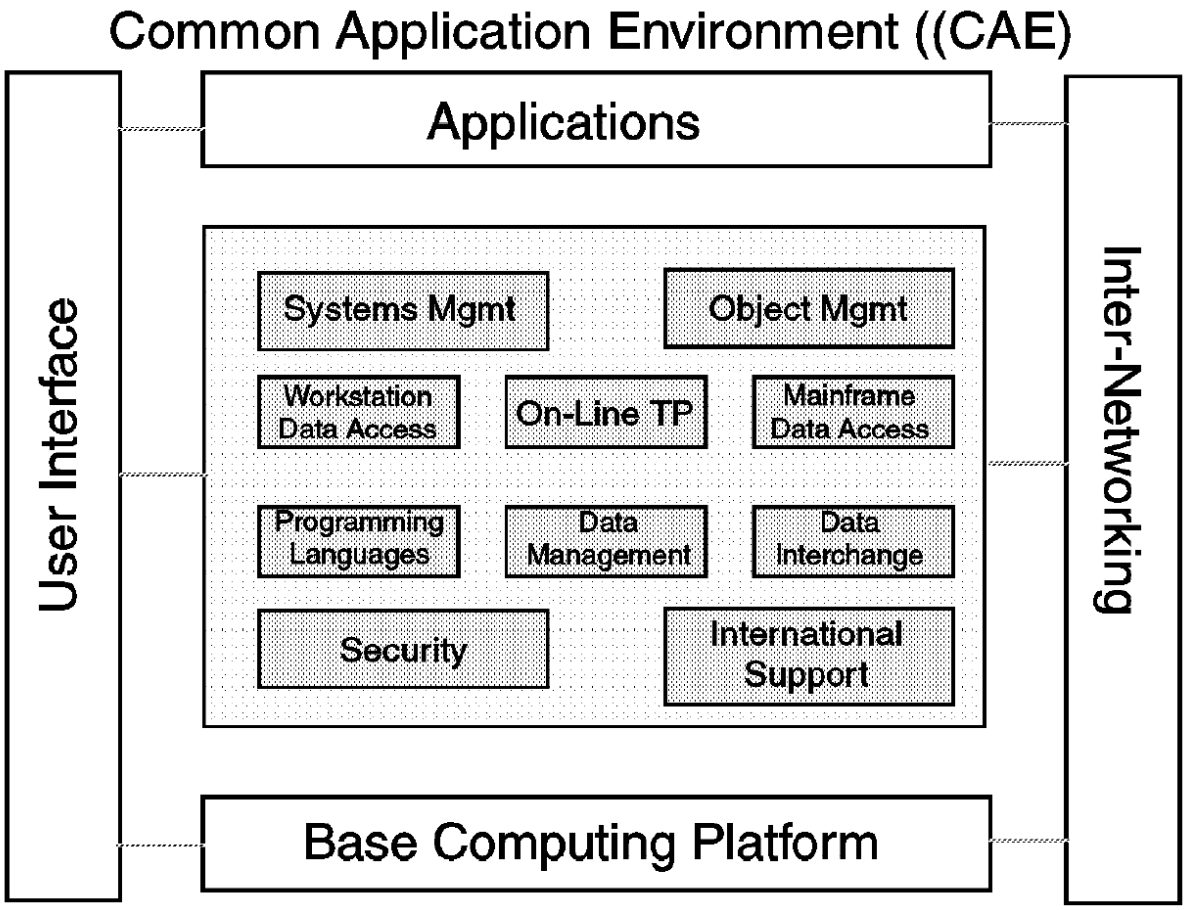
**früher als Open Edition MVS (OMVS) bezeichnet**

**1100 Unix API´s**

| | | |
|---|---|---|
| * | 1003.1 | System Application Program Interface for C |
| * | 1003.1a | System API Extensions |
| | 1003.1b | Real Time Extensions |
| * | 1003.1c | Threads Extensions (previous 1003.4a) |
| | 1003.1e | Security Extensions |
| | 1003.1f | Network - Transparent File Access |
| | 1003.1g | Protocol Independant Network API |
| * | 1003.2 | Shell and Utilities |
| * | 1003.5 | ADA Bindings (for 1003.1) |
| * | 1003.9 | Fortran Bindings (for 1003.1) |
| | 1003.13 | Real- Time Application Environment Profile |
| | 1003.15 | Batch System Administration |
| | 1003.* | |

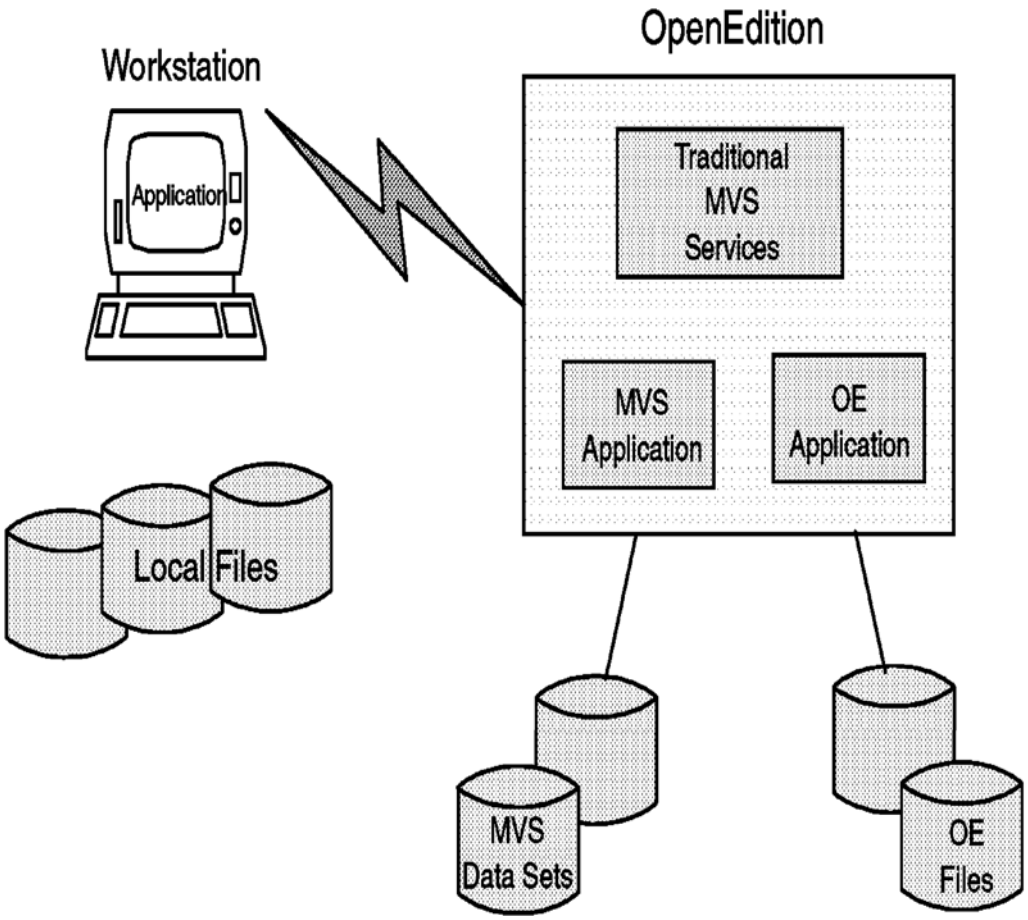
**The work on Portable Operating Systems Interface (Posix) started out as an effort to standardize UNIX and was performed by a work group under IEEE (Institute of Electrical and Electronic Engineers). What they defined was an application programming interface which could be applied not only to UNIX systems but to other operating systems, like OS/390.**

**Posix is now a set of standards that define common interfaces across operating systems. Most of the interfaces are defined in terms of C language. The Posix standard is sponsored by ISO (International Standards Organization) and is incorporated into the X/Open Portability Guides. Each element of the standard is defined by a 1003.* number.**




**X/Open is dedicated to developing an open, multivendor Common Applications Environment (CAE). Specification of the CAE is published in the X/Open Portability Guide (XPG) whose latest versions are referred to as XPG4 and XPG4.2.**

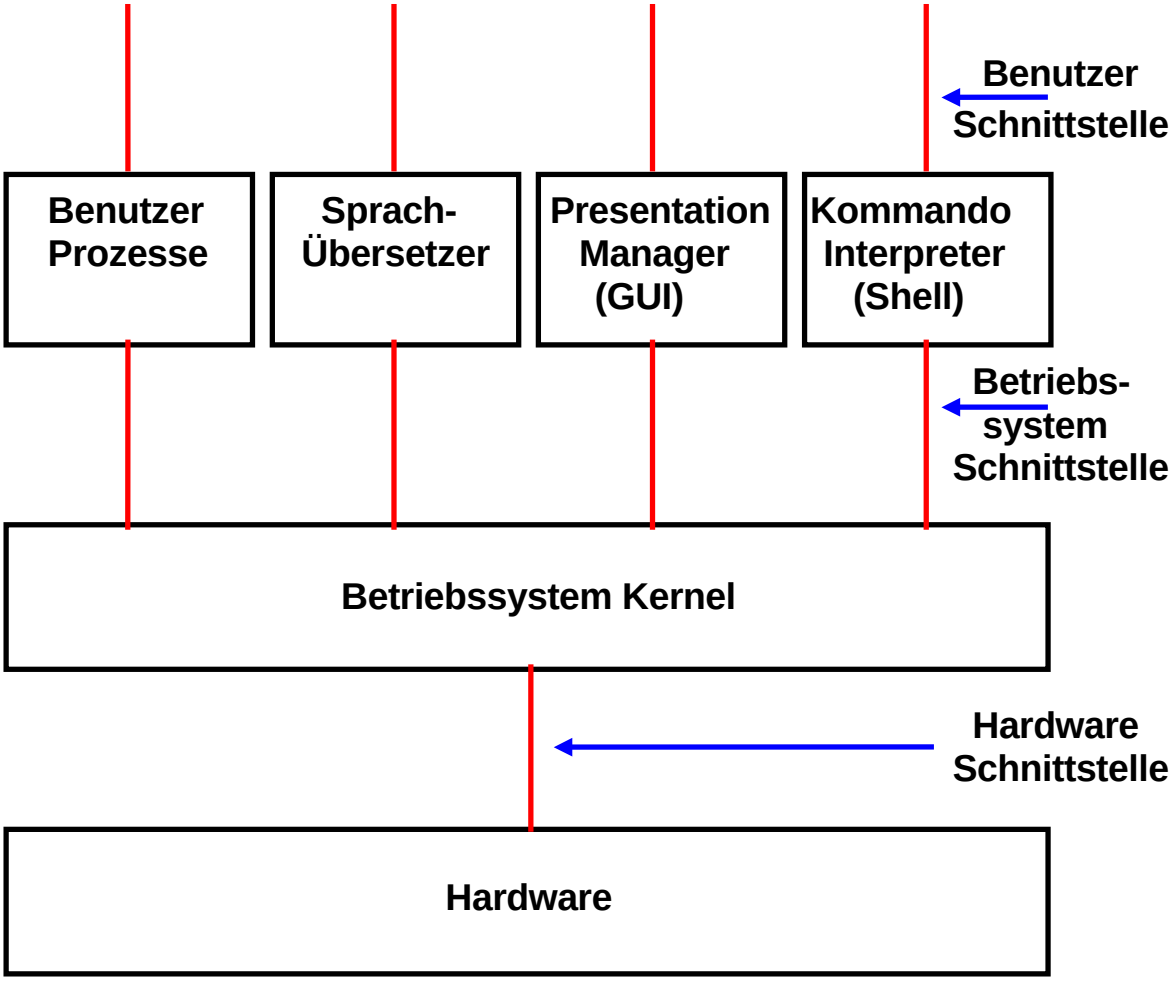
**The CAE is an open systems environment specification based upon both formal and de facto standards. The intent is to provide application portability and interoperability while allowing movement of users between systems without re-training. The foundation of CAE lies in the specification of the C language, in the Posix (1003.1 and 1003.2) standard, and AT&T System V Interface Definition (SVID).**



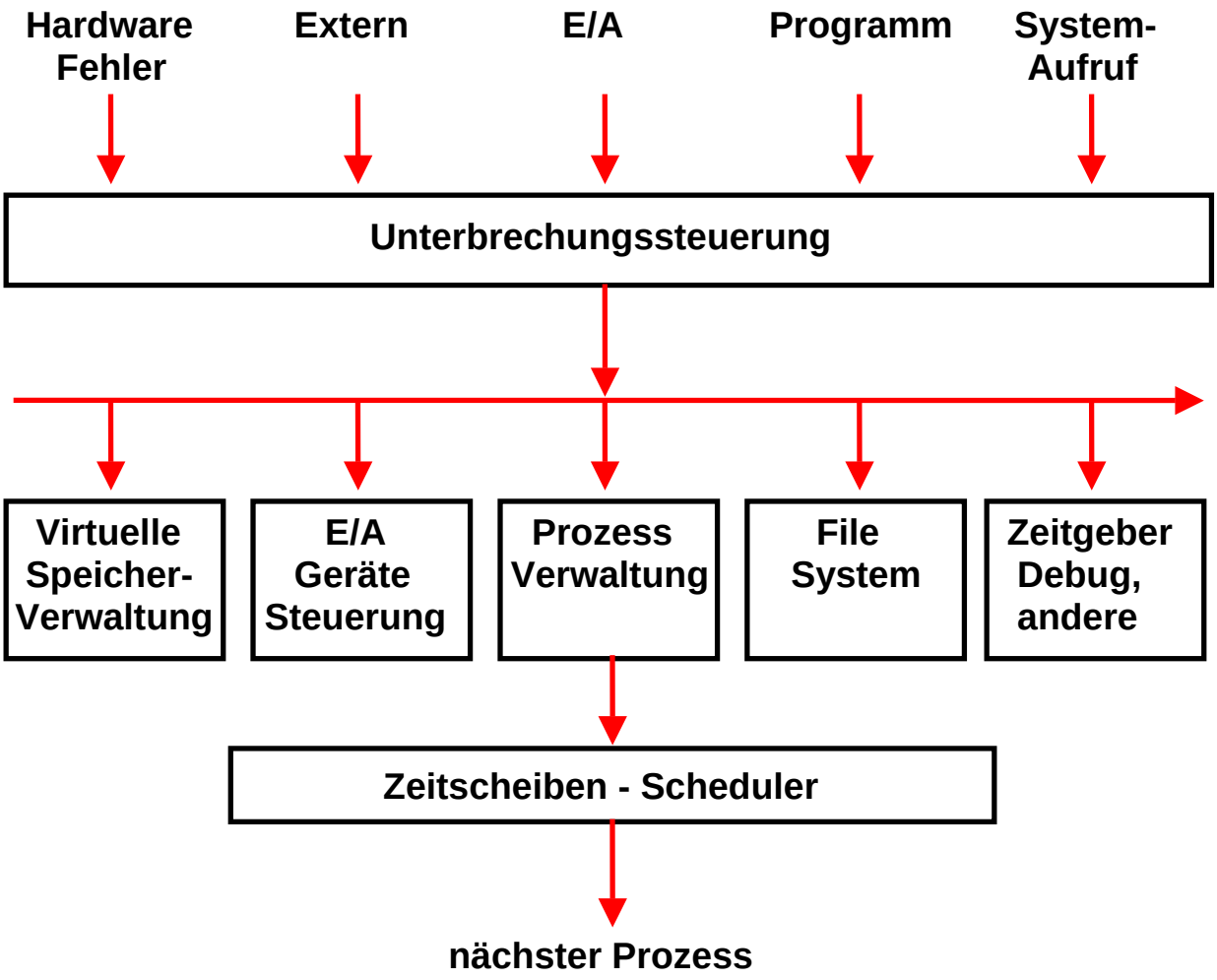

# Unix System Services

**OpenEdition**

**z/OS verhält sich entweder wie ein traditionelles MVS System oder wie ein Unix System. Letzteres wird durch eine Zusatz Einrichtung, den Unix System Services möglich. Frühere Bezeichnung : OpenEdition oder Open MVS.**




# Schichtenmodell der Rechnerarchitektur




# Struktur des Überwachers

**Der Ufruf des Überwachers (Supervisor, Kernel) erfplgt grundsätzlich über Unterbrechungen.**

**Je nach Art der Unterbrechung werden unterschiedliche Komponenten des Überwachers aufgerufen.**

**Der Scheduler (Zeitscheibensteuerung) sucht den nächsten auszuführenden Prozess aus.**

# Supervisor Calls

**SVC's (Supervisor Calls) sind das Äquivalent zu den Unix System Calls.**

**Supervisor Calls im weiteren Sinne sind Library Routinen, die eine Dienstleistung des OS/390 Kernels in anspruch nehmen.**

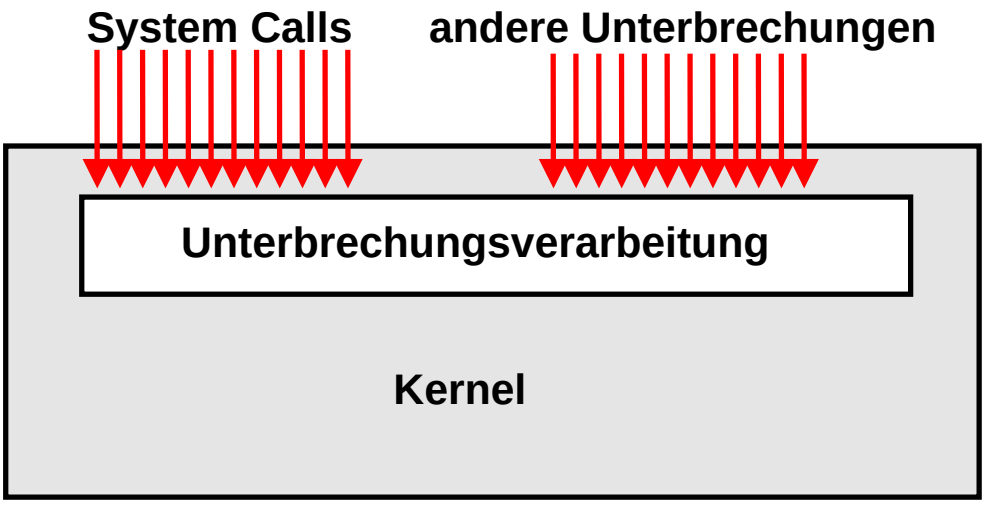
**Supervisor Calls im engeren Sinne sind Maschinenbefehle, die über einen Übergang vom User Mode (Problem Status) zum Kernel Mode (Supervisor Status) einen Interrupt Handler des Kernels aufrufen.**

**Ein SVC Maschinenbefehl enthält einen 8 Bit Identifier, welcher die Art des Supervisor Calls identifiziert.**

**Beispiele sind:**

| | | |
|---|---|---|
| **GETMAIN** | **SVC 10** | **Anforderung von Virtual Storage** |
| **OPEN** | **SVC 19** | **Öffnen eines Data Sets** |
| **EXCP** | **SVC 0** | **Lesen oder Schreiben von Daten** |
| **WAIT** | **SVC 19** | **Warten auf ein Ereignis, z.B. Abschluß einer Lese Operation** |

**Unix System Services sind eine Erweiterung des z/OS Kernels (BCP)um 1100 Unix System Calls, als zSeries SVCs implementiert.**




**Die Kernel aller Betriebssysteme haben de facto identische Funktionen, z. B.**

- **Unterbrechungsverarbeitung**
- **Prozessmanagement**
- **Scheduling/Dispatching**
- **Ein/Ausgabe Steuerung**
- **Virtuelle Speicherverwaltung**
- **Dateimanagement**

**Ein Unix Kernel unterscheidet sich von einem Windows Kernel durch die Syntax und Semantik der unterstützten System Calls, seine Shells sowie durch die unterstützen Dateisysteme.**

**Linux, Solaris, HP-UX und AIX haben unterschiedliche Kernel, aber (nahezu) identische System Calls..**

**Die Unix System Services (USS) des z/OS Betriebssystems sind eine Erweiterung des z/OS Kernels um 1100 Unix System Calls, zwei Unix Shells und zwei Unix Datei-systeme. Damit wird aus z/OS ein Unix Betriebssystem.**

# Microsoft Windows Services for UNIX

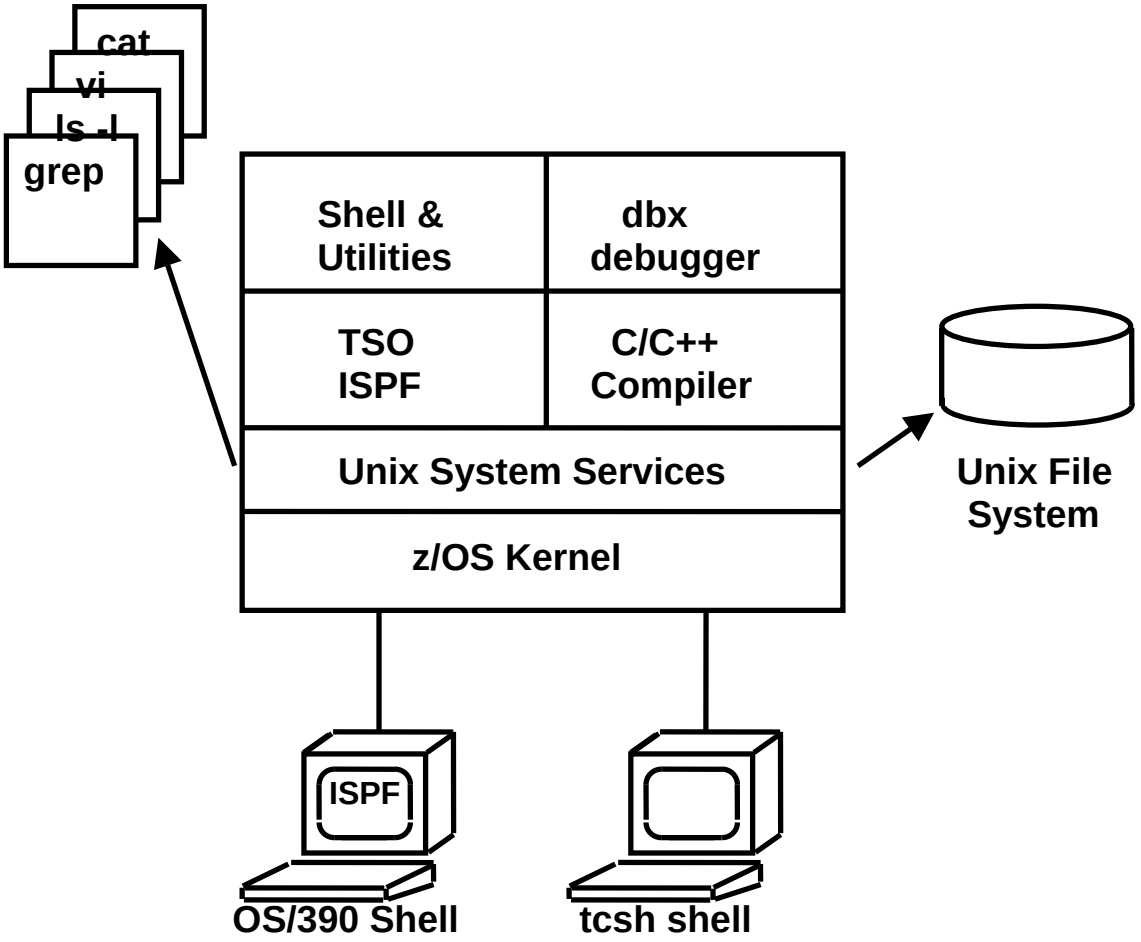
**Microsoft Windows Services for UNIX (SFU) ist ein Software-Paket von Microsoft, welches ähnlich wie z/OS Unix System Services arbeitet. Es handelt sich hierbei um ein Unix-Subsystem (und weitere Komponenten einer Unix-Umgebung) nach POSIX-Standard, welches unter den Windows Betriebssystemen verfügbar ist. Dieses Subsystem wird auch als Interix bezeichnet.**

**Microsoft Windows Services for UNIX setzt als Implementierung eines User-Mode-Subsystems unmittelbar auf dem Windows-XP-Kernel auf.**

**Die aktuelle Version trägt die Nummer 3.5. Als Veröffentlichungsdatum wird der 21. September 2006 angegeben.**

**Microsoft Windows Services for UNIX kann auf den folgenden Windows-Varianten installiert werden:**

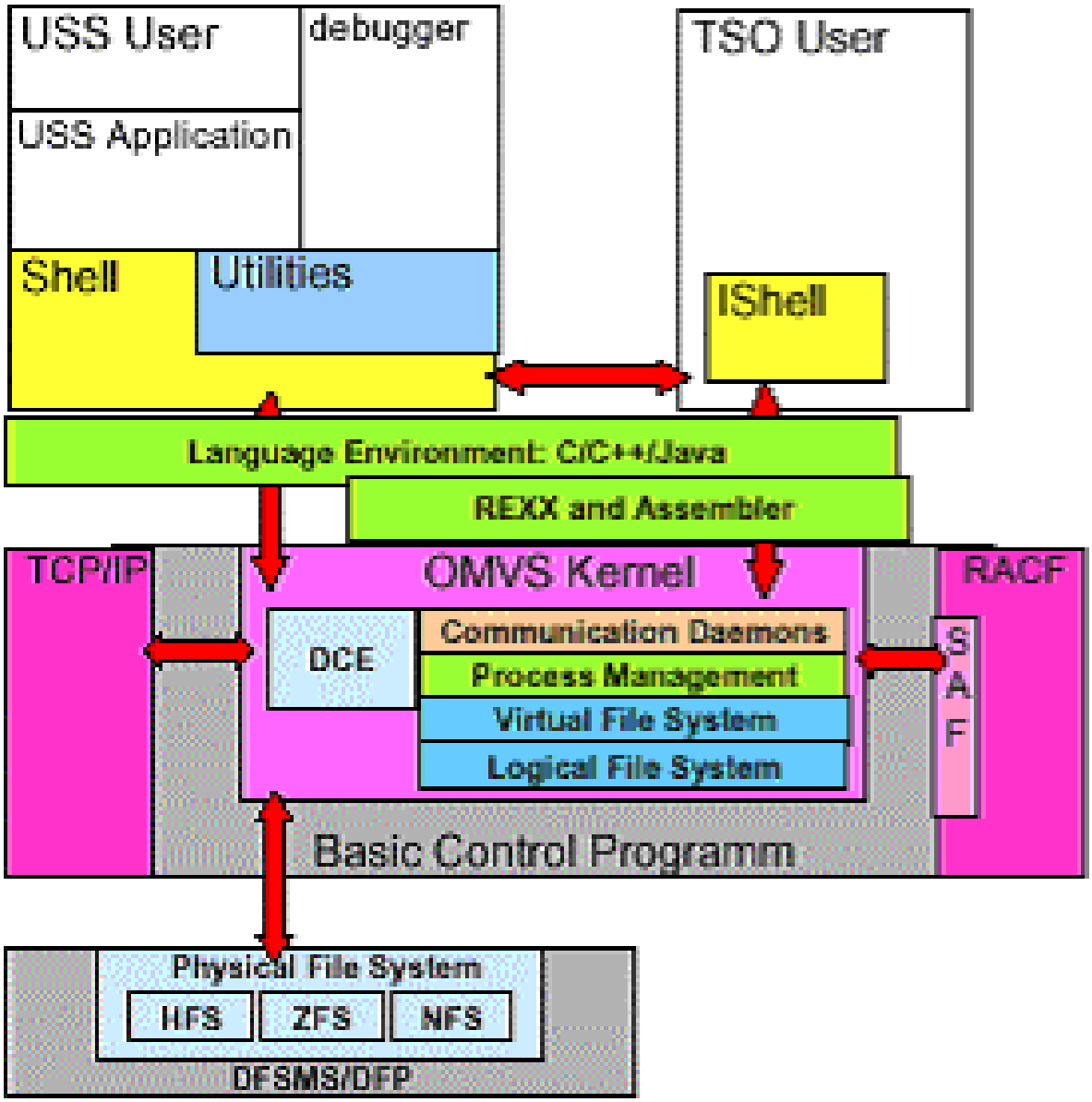
- **Windows 2000 Server oder Professional mit Service Pack 3 oder neuer**
- **Windows XP Professional mit Service Pack 1 oder neuer**
- **Windows Server 2003**
- **Windows Vista.**




**z/OS Unix System Services verfügt über zwei unterschiedliche Shells.**

**Die OS/390 Shell gleicht der Unix System V Shell mit einigen zusätzlichen Eigenschaften der Korn Shell. Sie wird meistens von TSO aus über das OMVS Kommando aufgerufen und benutzt den ISPF Editor.**

**Die tcsh Shell ist kompatibel mit der csh Shell, der Berkley Unix C Shell. Sie wird über rlogin oder telnet aufgerufen und verwendet den vi Editor.**

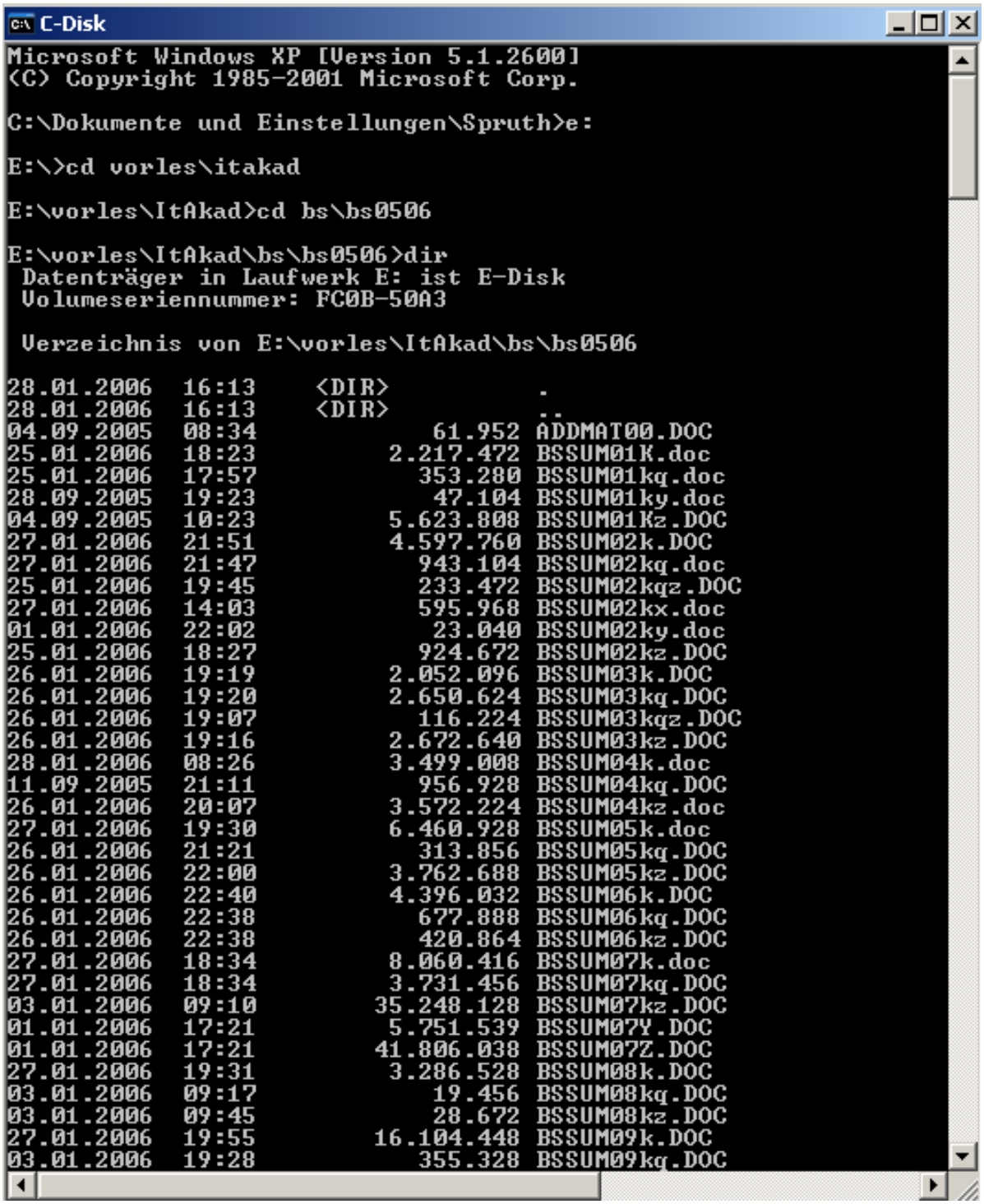



**Unix System Services (USS) stellt eine vollständige UNIX Umgebung unter OS/390 und z/OS zur Verfügung:**

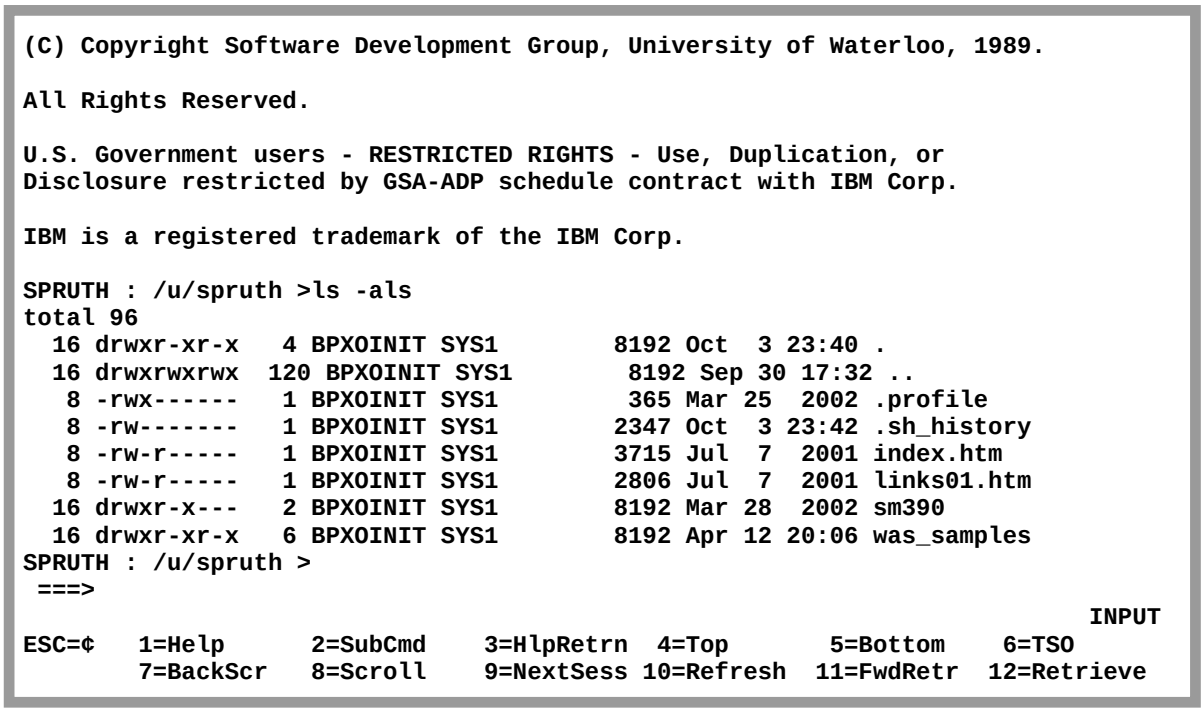
- **Posix 1003.2**
- **XPG/4: X/Open Portability Guide**

**Der OMVS Kernel ist Teil des z/OS bzw. OS/390 BCP (Kernel)**
**Mehrere Unix Shells: telnet, rlogin, 3270 basierte Interface, außerdem Zugriff über TSO auf USS.**
**File Systeme:**

- **HFS und zFS**
- **NFS (Network File System)**
- **DFS (Distributed File System, Teil von DCE)**




# DOS Shell
# des Windows Betriebssystems




# z/OS Unix System Services Shell

**z/OS Unix System Services verfügt über zwei unterschiedliche Shells.**

**Die OS/390 Shell gleicht der Unix System V Shell mit einigen zusätzlichen Eigenschaften der Korn Shell. Sie wird meistens von TSO aus über das OMVS Kommando aufgerufen und benutzt den ISPF Editor.**

**Die tcsh Shell ist kompatibel mit der csh Shell, der Berkley Unix C Shell. Sie wird über rlogin oder telnet aufgerufen und verwendet den vi Editor.**

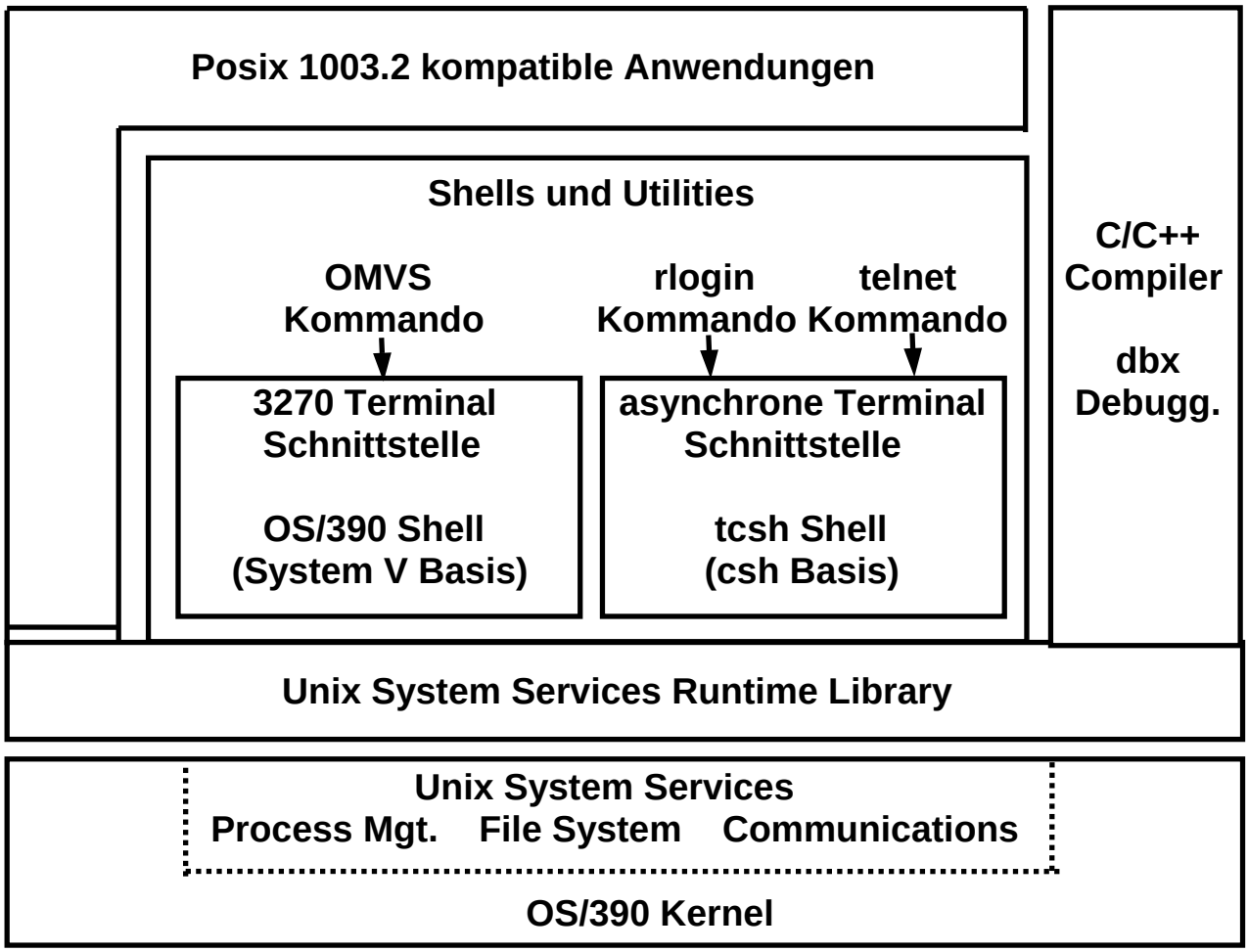
```
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

SPRUTH : /u/spruth >ls -als
total 96
  16 drwxr-xr-x   4 BPXOINIT SYS1        8192 Oct  3 23:40 .
  16 drwxrwxrwx  120 BPXOINIT SYS1        8192 Sep 30 17:32 ..
   8 -rwx------   1 BPXOINIT SYS1         365 Mar 25  2002 .profile
   8 -rw-------   1 BPXOINIT SYS1        2347 Oct  3 23:42 .sh_history
   8 -rw-r-----   1 BPXOINIT SYS1        3715 Jul  7  2001 index.htm
   8 -rw-r-----   1 BPXOINIT SYS1        2806 Jul  7  2001 links01.htm
  16 drwxr-x---   2 BPXOINIT SYS1        8192 Mar 28  2002 sm390
  16 drwxr-xr-x   6 BPXOINIT SYS1        8192 Apr 12 20:06 was_samples
SPRUTH : /u/spruth >
 ===>
                                                                      INPUT
ESC=¢   1=Help      2=SubCmd    3=HlpRetrn  4=Top       5=Bottom    6=TSO
        7=BackScr   8=Scroll    9=NextSess 10=Refresh  11=FwdRetr  12=Retrieve
```




# Unix System Services (USS)

**OS/390 Unix System Services Kernel Funktion läuft in einem eigenen virtuellen Adressenraum, der als Teil des IPL Vorgangs hochgefahren wird. Er wird wie jeder Unix Kernel über eine API aufgerufen, die aus C/C++ Function Calls besteht.**

**Das Byte-orientierte hierarchische File System arbeitet wie jedes Unix File System. Es wird in OS/390 Data Sets abgebildet. Alle Files sind dem Unix System Services Kernel zugeordnet, und alle Ein-/Ausgabe Operationen bewirken Calls in den Kernel.**

**Die tsch Shell wird normalerweise über rlogin aufgerufen.**

**Es ist nicht unüblich, auf dem gleichen S/390 Rechner Unix System Services und S/390 Linux parallel zu betreiben.**

z/OS UNIX
(z/OS Shell)
OMVS command

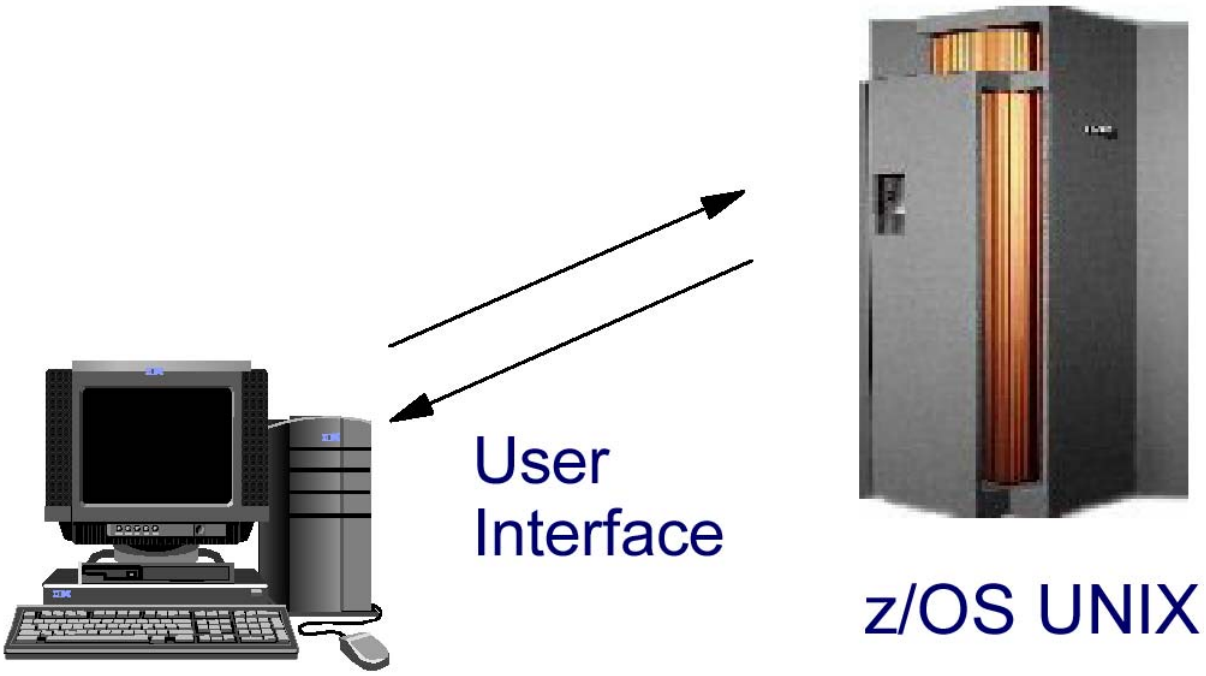

- UNIX interface
- POSIX 1003.2
- Command interface

UNIX experienced user

ISPF Shell
(ISHELL)
ishell command


- ISPF based
- Menu interface

TSO experienced user

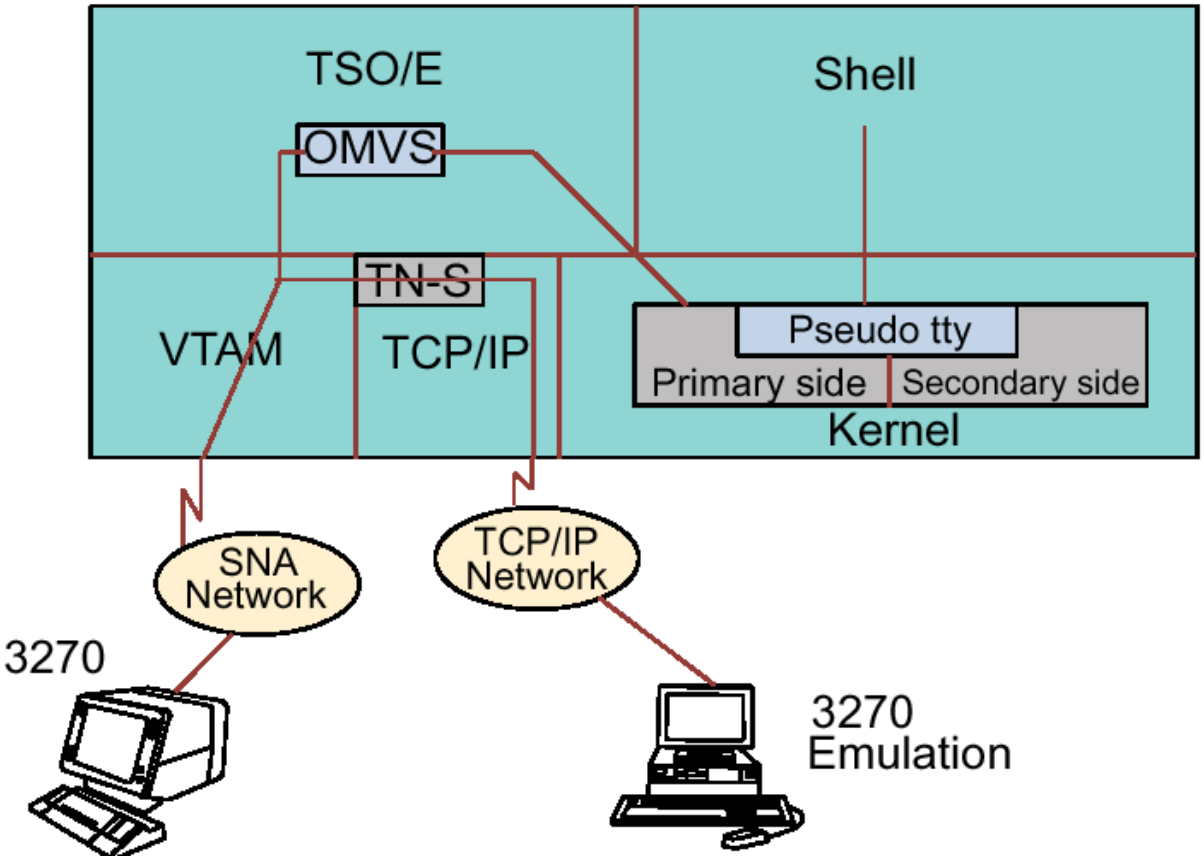



# TSO and z/OS UNIX

**One way to enter the UNIX shell environment is by using TSO/E. A user logs on to a TSO/E session and enters the TSO/E OMVS command.**

**The z/OS environment has other TSO/E commands, for example, to logically mount and unmount file systems, create directories in a file system, and copy files to and from MVS data sets. Users can switch from the shell to their TSO/E session, enter commands, or do editing, and switch back to the shell.**


Menu List Mode Functions Utilities Help
ISPF Command Shell
Enter TSO or Workstation commands below:
===> OMVS
Place cursor on choice and press enter to Retrieve command
=>
=>
=>
=>
=>
=>
=>
=>
=>
=>
F1=Help F3=Exit F10=Actions F12=Cancel

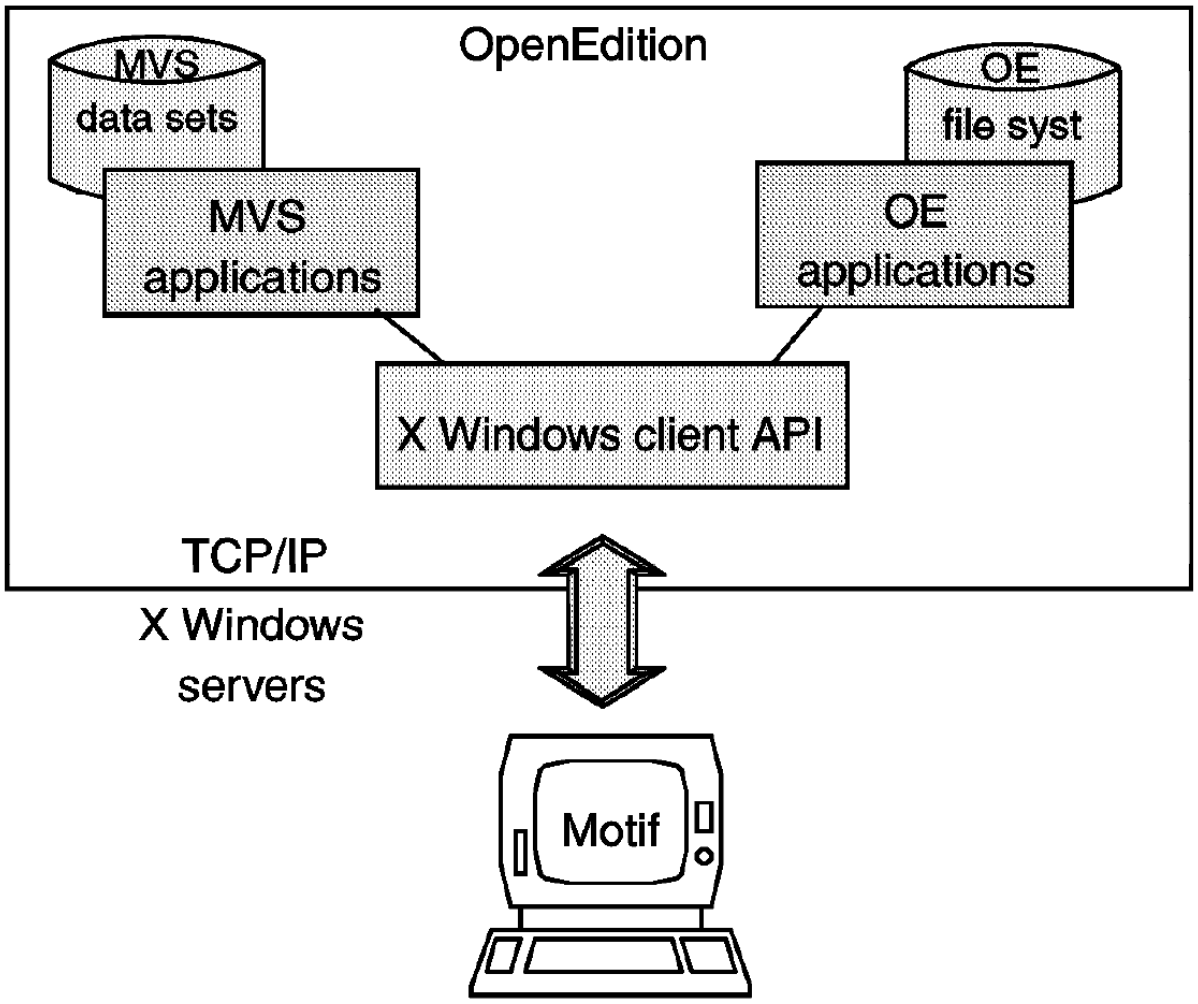



## TSO/E Shell-Zugriff

**Um interaktiv mit dem Unix arbeiten zu können und um Programme starten zu können, ist ein Shell-Zugriff nötig. Insgesamt drei Optionen für einen Shell-Zugriff stehen zur Verfügung:**

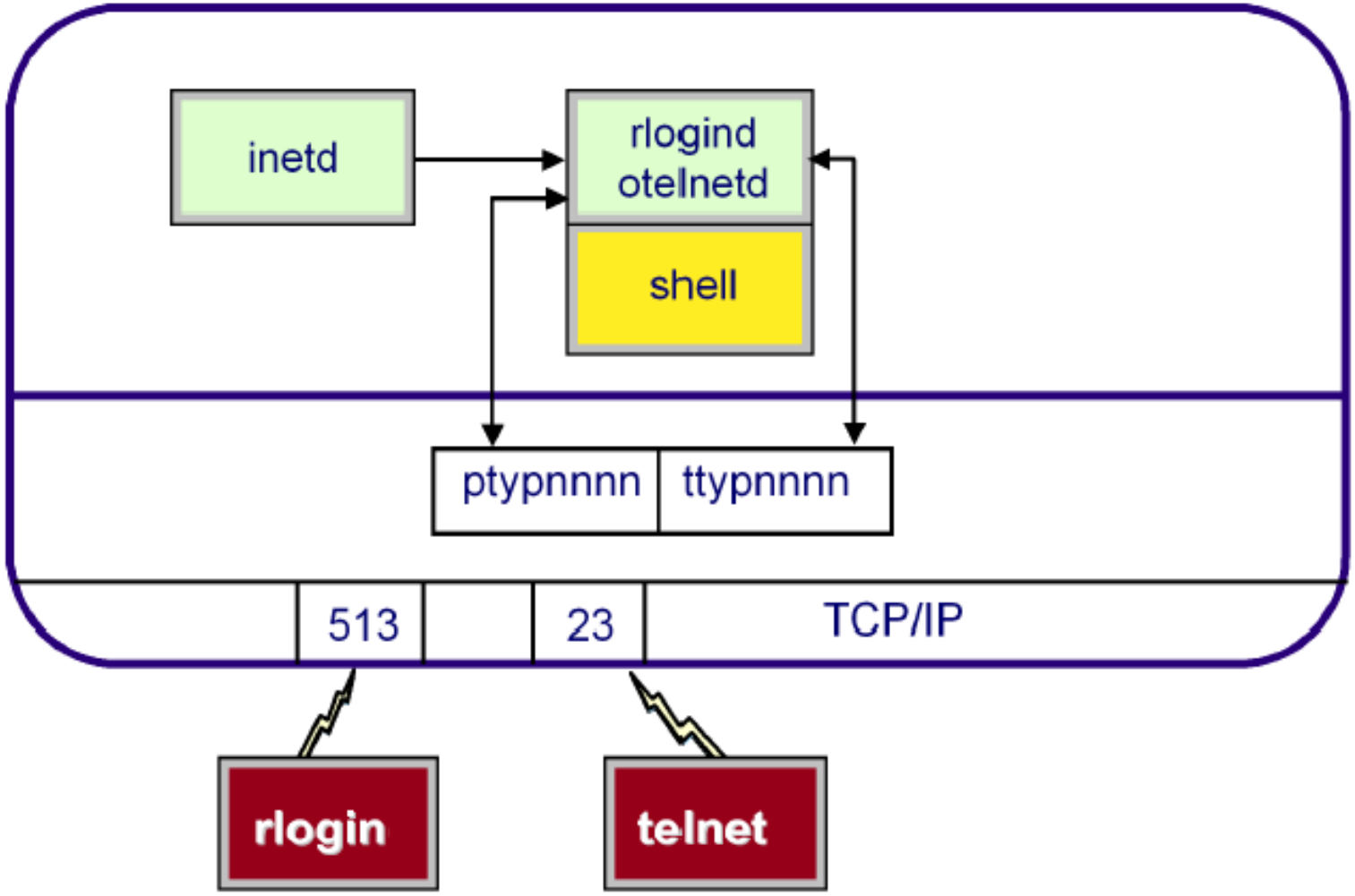
1. **Um reibungslos aus einem vorhandenen SNA-Netzwerk Aufschlag die Shell zugreifen zu können, ist der TSO-Command- Processort OMVS neu eingeführt worden. Die Grafik stellt zwei Optionen dieses Zugriffes dar.**
2. **Das breite Batch-Spektrum eines OS/390 erforderte eine weitere Zugriffsmöglichkeit. Mit dem Aufruf //S1 EXEC PGM=BPXBATCH,PARM='SH /bin/pax ... ' können Utilities und Shell-Scripte aus einem Batch-Job heraus gestartet werden.**
3. **Die klassische Unix-Form des Shell-Zugriffes über telnet oder rlogin ist in einem TCP/IP-Netz möglich.**

**Zu beachten ist, dass der Standard-Telnet- Server des OS/390 TCP/IP als sogenannter TN3270E-Server den Port 23 benutzt.**



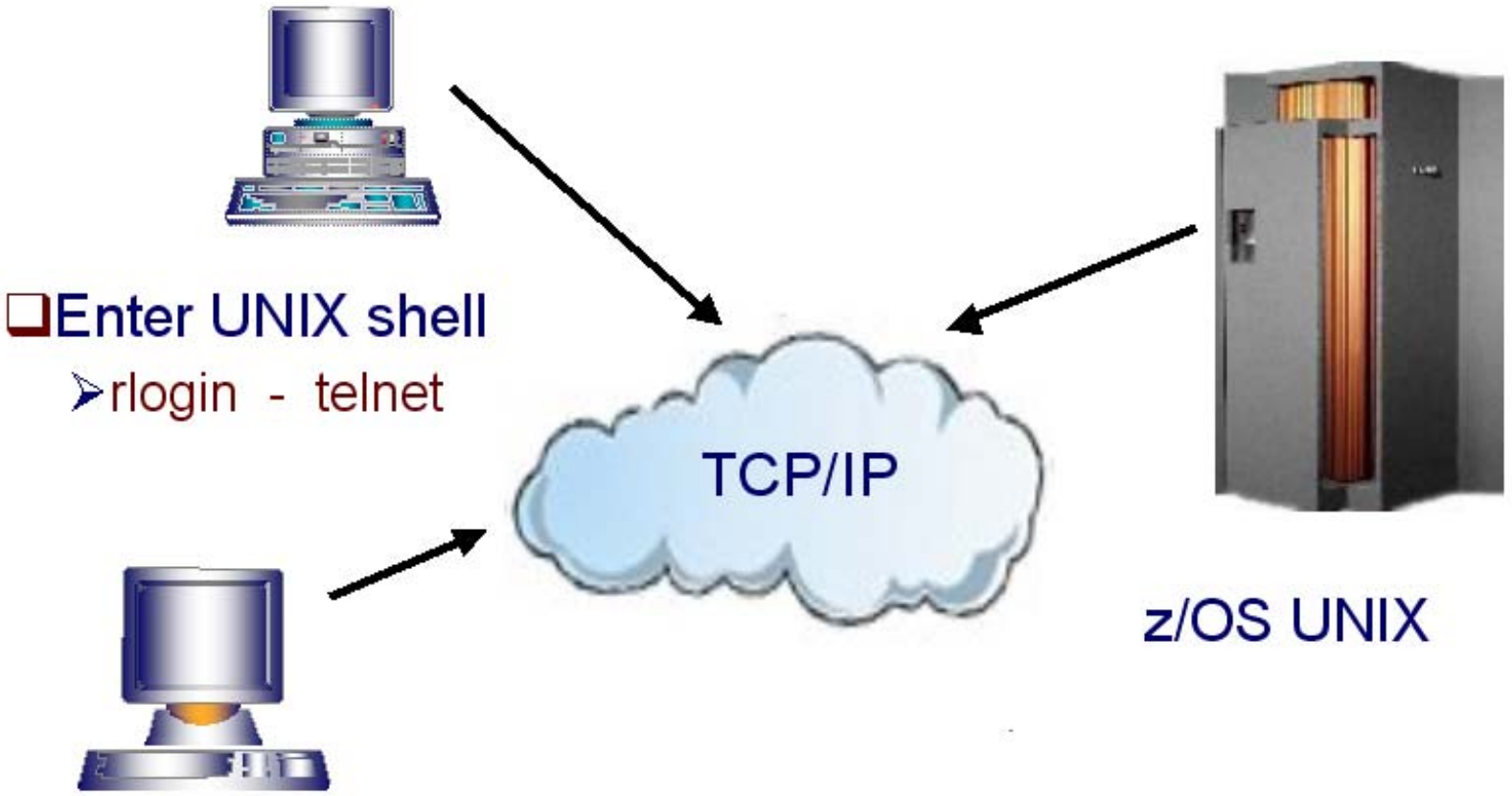

**Mit Unix System Services können Arbeitsplatzrechner (Workstations) auf traditionellle z/OS Anwendungen und Daten zugreifen. Ebenso können sie unter z/OS auf Anwendungen und Daten zugreifen, die den Posix und XPG4/4.2 Standards entsprechen.**

**Wenn auf der Workstation X Windows installiert ist, können Benutzer auf z/OS und XPG Anwendungen über TCP/IP unter Nutzung der Motif Graphical User Interface (GUI) zugreifen. Die Anwendung auf der Workstation muss hierzu die X Windows Server Request Funktion unterstützen.**

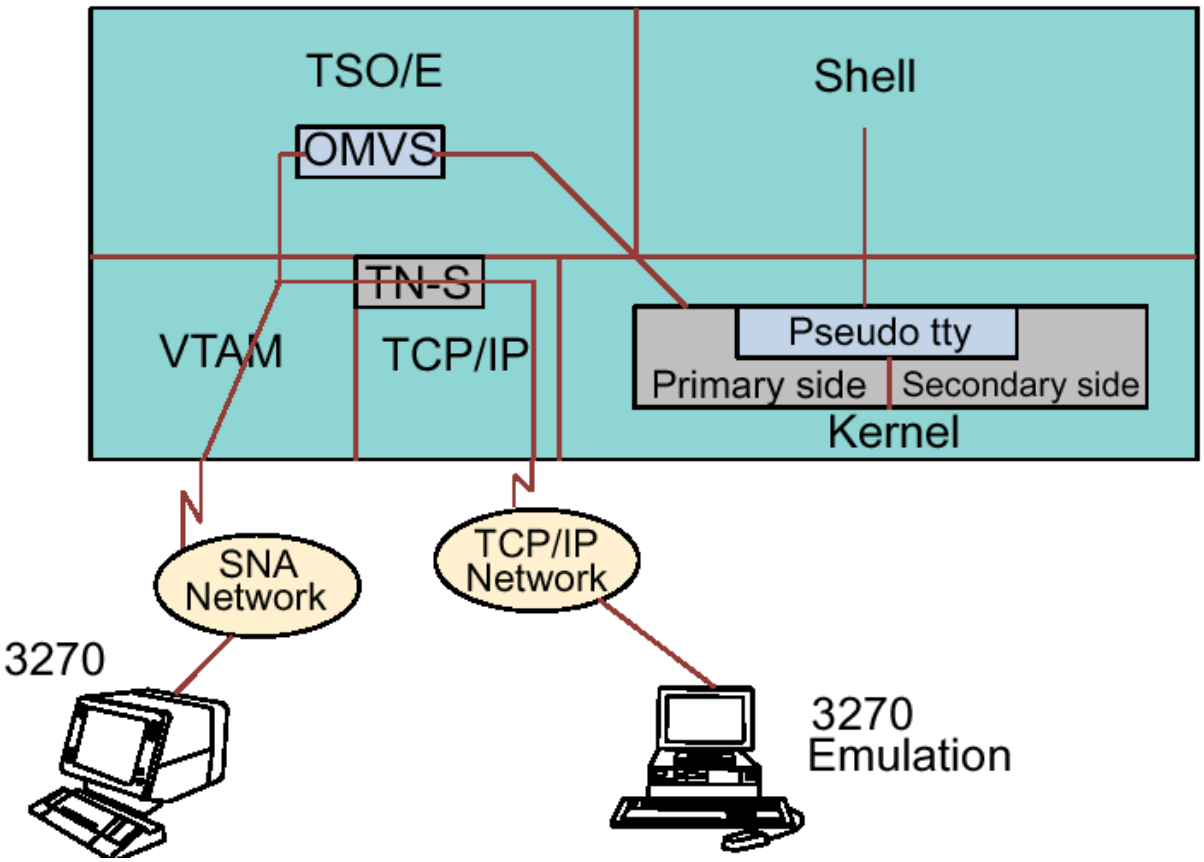



**Aufruf der Shell über rlogin oder telnet**

# Transmission Control Protocol Internet Protocol (TCP/IP)




- Enter UNIX shell
  - rlogin - telnet
- User written socket applications

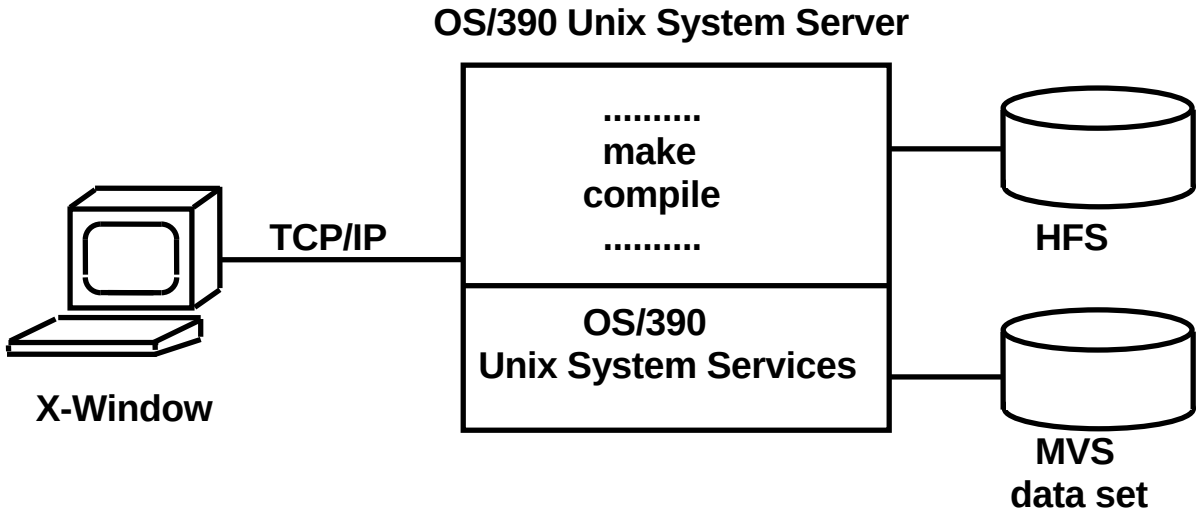



## TSO/E Shell-Zugriff

**Um interaktiv mit dem Unix arbeiten zu können und um Programme starten zu können, ist ein Shell-Zugriff nötig. Insgesamt drei Optionen für einen Shell-Zugriff stehen zur Verfügung:**

4. **Um reibungslos aus einem vorhandenen SNA-Netzwerk Aufschlag die Shell zugreifen zu können, ist der TSO-Command- Processort OMVS neu eingeführt worden. Die Grafik stellt zwei Optionen dieses Zugriffes dar.**
5. **Das breite Batch-Spektrum eines OS/390 erforderte eine weitere Zugriffsmöglichkeit. Mit dem Aufruf //S1 EXEC PGM=BPXBATCH,PARM='SH /bin/pax ... ' können Utilities und Shell-Scripte aus einem Batch-Job heraus gestartet werden.**
6. **Die klassische Unix-Form des Shell-Zugriffes über telnet oder rlogin ist in einem TCP/IP-Netz möglich.**

**Zu beachten ist, dass der Standard-Telnet- Server des OS/390 TCP/IP als sogenannter TN3270E-Server den Port 23 benutzt.**




# Compiling with c89

**Application source code in HFS files or OS/390 data sets can be compiled and link-edited from within the shell using the Unix System Services c89 utility. The syntax is:**

**c89 [-options ...] [file.c ...] [file.a ...] [file.o ...] [-l libname]**

**where:**

| | |
|---|---|
| **file.c** | **source file** |
| **file.o** | **object file** |
| **file.a** | **archive file** |
| **libname** | **archive library** |

**Makefiles may be used to maintain Unix System Services application source and object files automatically when updating individual modules. The make utility runs c89.**

**The X Window API allows to write applications in the OpenEdition MVS environment that can be displayed on X11 servers on a TCP/IP based network. It provides that application with graphics capabilities as defined by the X Window System protocol.**

## ASCII-Tabelle

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS | HT | LF | VT | FF | CR | SO | SI |
| **1** | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
| **2** | SP | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| **3** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| **4** | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| **5** | P | Q | R | S | T | U | V | W | X | Y | Z | Ä | Ö | Ü | ^ | _ |
| **6** | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| **7** | p | q | r | s | t | u | v | w | x | y | z | ä | | | | |

## EBCDIC-Tabelle

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | NUL | SOH | STX | ETX | PF | HT | LC | DEL | | | SMM | VT | FF | CR | SO | SI |
| **1** | DLE | DC1 | DC2 | TM | RES | NL | BS | IL | CAN | EM | CC | CU1 | IFS | IGS | IRS | IUS |
| **2** | DS | SOS | FS | | BYP | LF | ETB | ESC | | | SM | CU2 | | ENQ | ACK | BEL |
| **3** | | | SYN | | PN | RS | UC | EOT | | | | CU3 | DC4 | NAK | | SUB |
| **4** | SP | | | | | | | | | | ¢ | . | < | ( | + | \| |
| **5** | & | | | | | | | | | | ! | $ | * | ) | ; | ¬ |
| **6** | - | / | | | | | | | | | | , | % | _ | > | ? |
| **7** | | | | | | | | | | | : | # | @ | ' | = | " |
| **8** | | a | b | c | d | e | f | g | h | i | | | | | | |
| **9** | | j | k | l | m | n | o | p | q | r | | | | | | |
| **A** | | ~ | s | t | u | v | w | x | y | z | | | | | | |
| **B** | | | | | | | | | | ` | | | | | | |
| **C** | | A | B | C | D | E | F | G | H | I | | | | | | |
| **D** | | J | K | L | M | N | O | P | Q | R | | | | | | |
| **E** | \ | | S | T | U | V | W | X | Y | Z | | | | | | |
| **F** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | | | | | |

# Pitfalls

**The C language allows to code the following:**

```
char z;
z = '7' ;
if (47 < z && z < 58) { /* is numeric */
some code here
}
```

**What the code does is to check if the binary representation of z lies in the region between decimal 47 and 58. If you look up an ASCII table you will find that the numbers 0-9 are located at these code points. This code will compile immediately without any error on every machine.**

**What happens if such a piece of code is ported to the Unix System Services environment and translated to EBCDIC? It will definitely fail. Thoughtful testing is required even if the source code compiles immediately.**

**The OS/390 C compiler provides services which makes such a piece of code portable across ASCII and EBCDIC platforms**

**A similar problem occurs because the ASCII characters are stored contiguously in the code table, where the EBCDIC characters are not. The following do-loop runs nicely and makes sense in an ASCII environment.**

```
for (i='A';i<='Z';i++)  do something here }
```

**As the alphabet in an EBCDIC environment is not stored contiguously this piece of code is not directly portable to Unix system Services**

# ASCII – EBCDIC conversion

**The libascii functions are integrated into the base of the Language Environment. They help you port ASCII-based C applications to the EBCDIC-based z/OS UNIX environment.**

**The C/C++ run-time library functions support EBCDIC characters. The libascii package provides an ASCII interface layer for some of the more commonly used C/C++ run-time library functions. libascii supports ASCII input and output characters by performing the necessary iconv() translations before and after invoking the C/C++ run-time library functions. Note that not all C functions are supported (see Limitations for additional information).**

**The XL C/C++ compiler predefined macro __STRING_CODE_SET__="ISO8859-1" generates ASCII characters rather than the default EBCDIC characters. Using this with the libascii code provides an ASCII-like environment.**

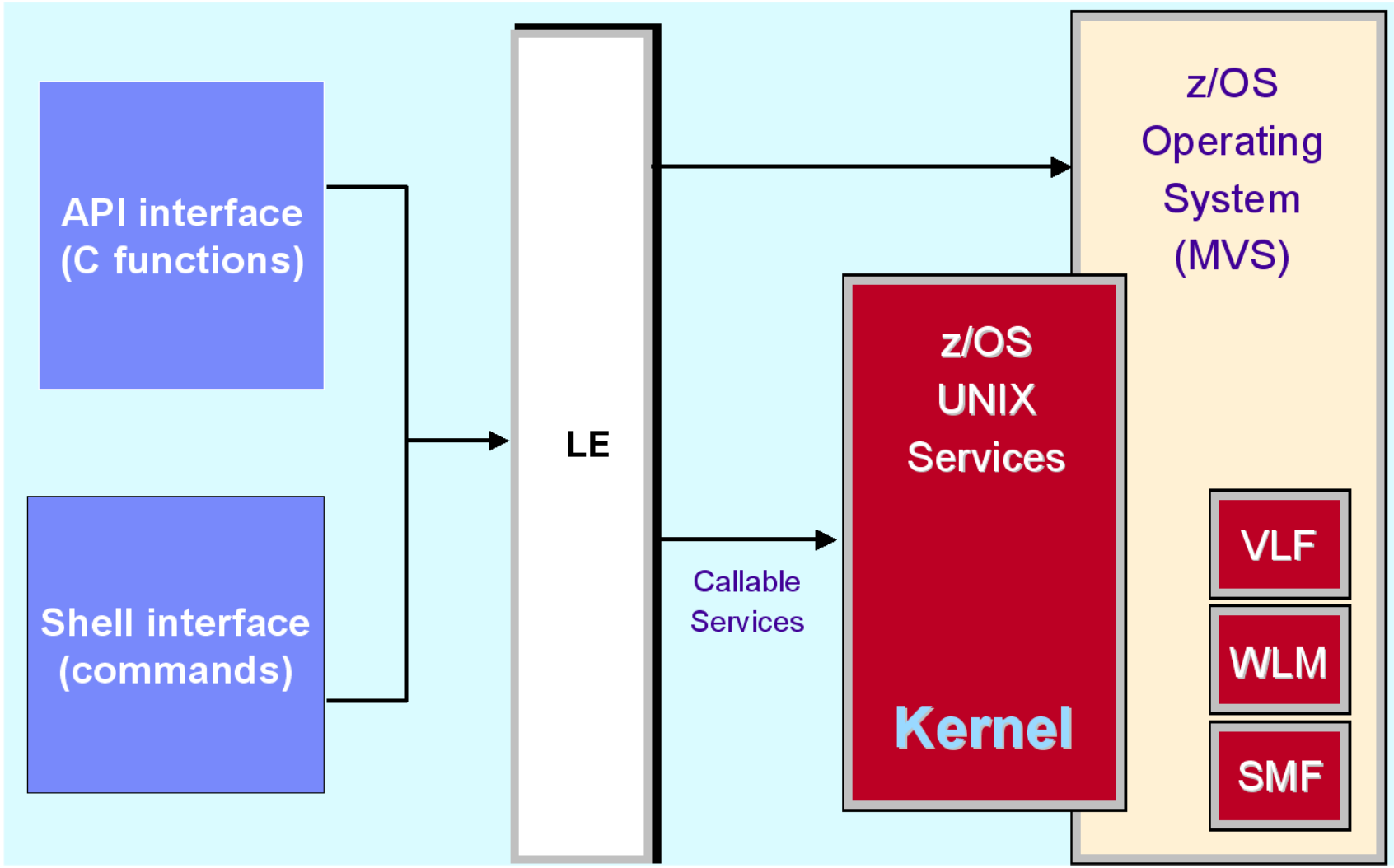
**The libascii package is as thread-safe as the run-time library except where stated under Limitations below.**

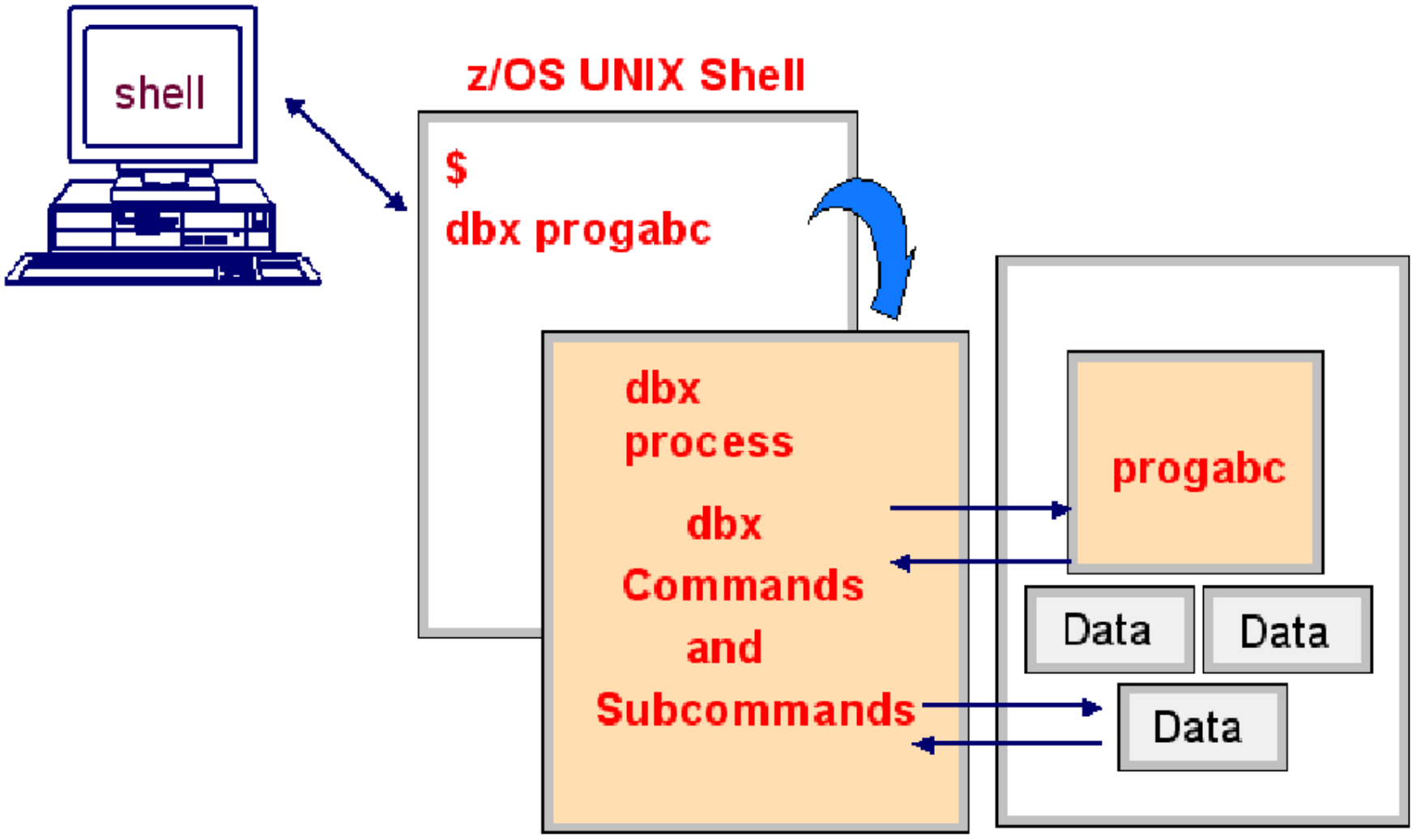
**You can download the package. The libascii source code is licensed by IBM; for more information, see the readme file in the package.**

**We have instructions on downloading through your browser and anonymous FTP.**

## Floating point conversion

**The libascii archive also contains four functions to convert between IEEE floating point and S/390 native hex floating point. The XL C/C++ compiler does not support IEEE floating point. Math operators such as + (add) and / (divide) and run-time library functions such as printf() and sin() using IEEE floating point numbers will produce undefined results. The main difference between the two floating point formats is IEEE supports a larger range of numbers, up to 10 to the 308 power. S/390 supports better precision but a smaller range, up to 10 to the 75 power.**


API interface
(C functions)
Shell interface
(commands)
LE
Callable
Services
z/OS
Operating
System
(MVS)
z/OS
UNIX
Services
Kernel
VLF
WLM
SMF




dbx Debugger

# z/OS UNIX dbx debugger

**The z/OS UNIX dbx debugger is an interactive tool for debugging C language programs that use z/OS UNIX. It is based upon the dbx debugger, which is regarded as an industry standard on UNIX systems.**

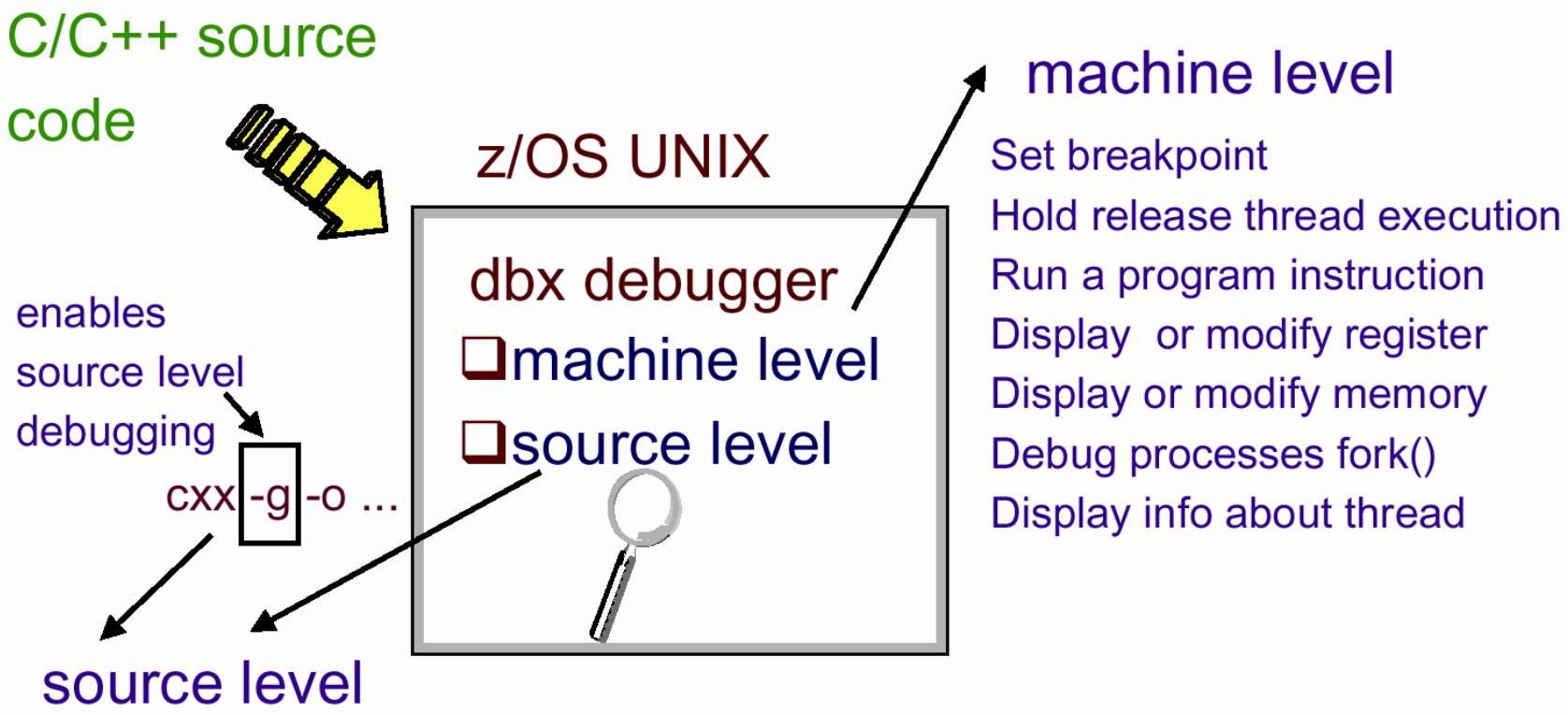
**The dbx debugger provides the options to debug at source level or assembler level. Source-level debugging allows you to debug C language programs. Assembler-level debugging allows you to debug executable programs at machine code level.**

**The dbx debugger is a utility that is invoked from the z/OS UNIX shell. It cannot be invoked directly from TSO/E. In the shell, dbx is the debugging facility for z/OS C/C++ programs. With dbx, you can debug multi-threaded applications at the C-source level or at the machine level.**

**Support for multi-threaded applications gives you the ability to:**

- **Debug or display information about the following objects related to multithreaded applications: threads, mutexes, and condition variables.**
- **Control program execution by holding and releasing individual threads.**

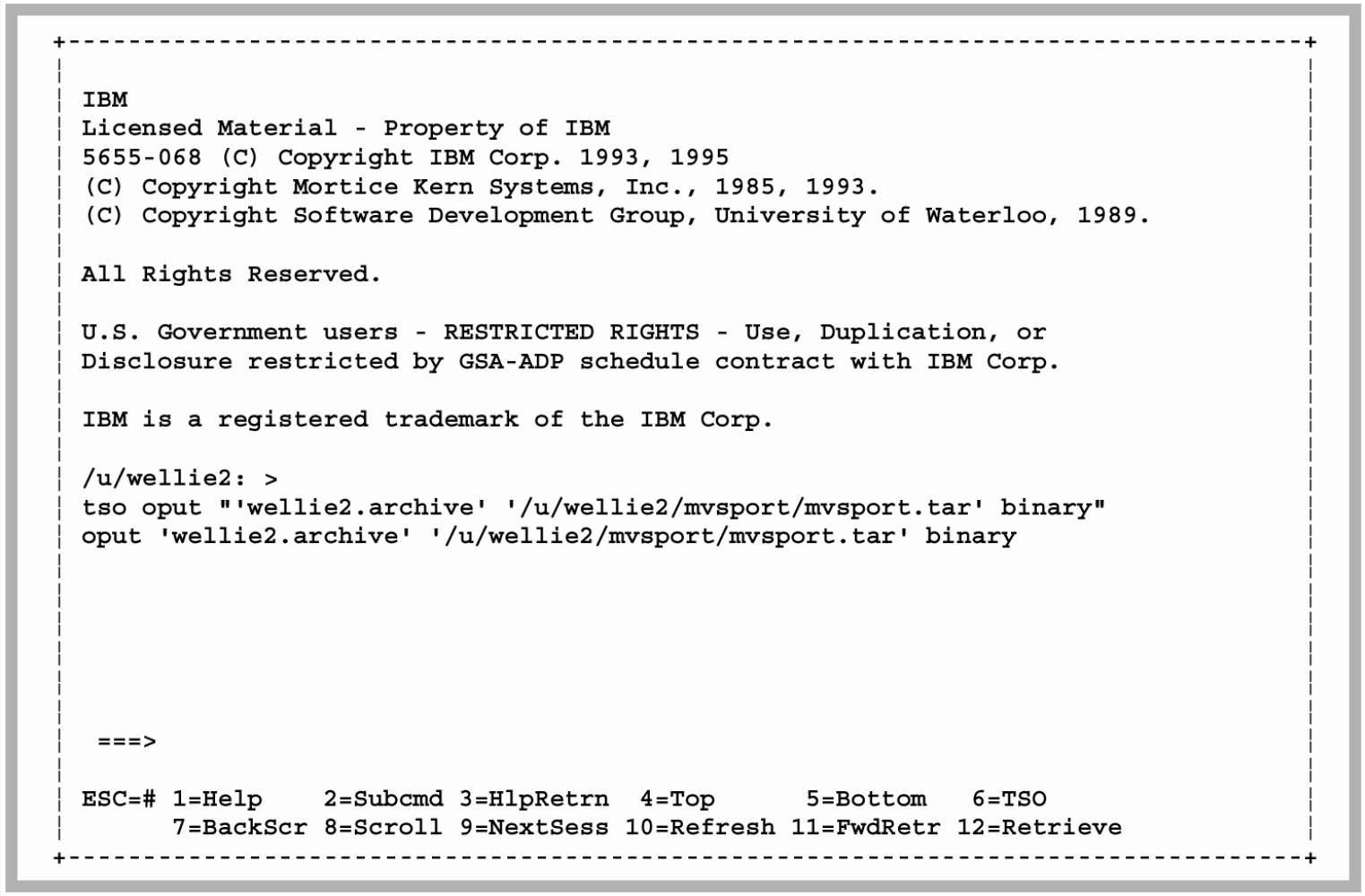
# dbx Debugger



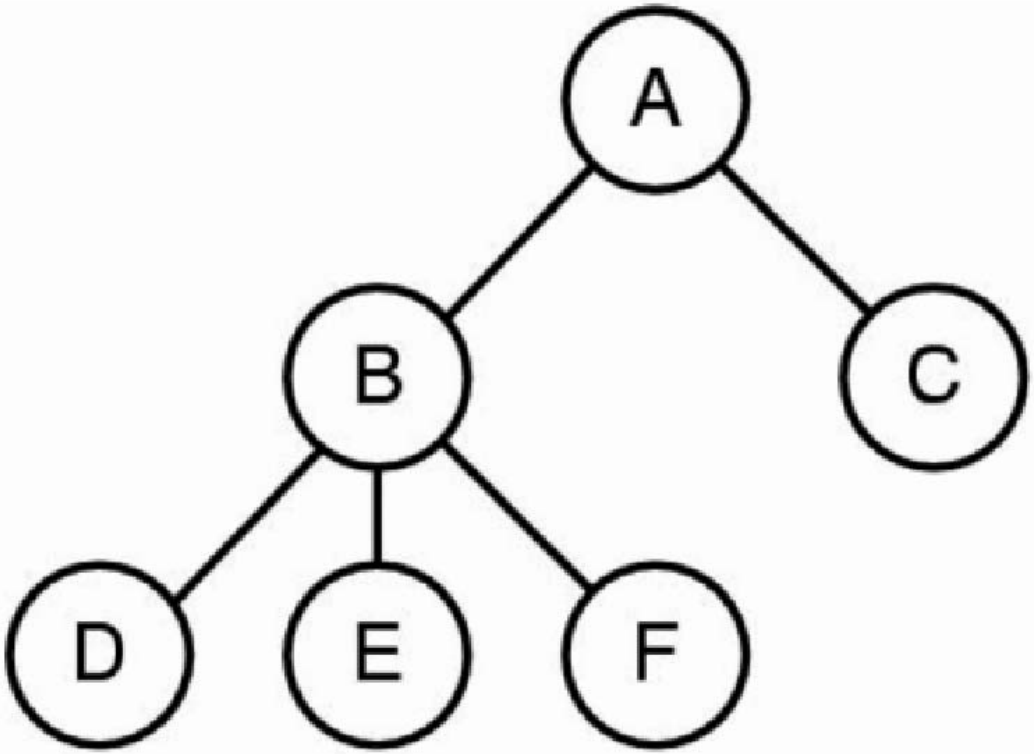
# Facilities of the interactive dbx debugger

**When using the interactive dbx debugger you can:**

- **Run a program one line or one instruction at a time.**
- **Set breakpoints at selected statements and machine instructions with conditions for activation.**
- **Access variables symbolically and display them in the correct format.**
- **Display or modify the contents of registers, variables, and storage.**
- **Examine the source text using simple search functions.**
- **Debug processes that contain fork() and exec() functions.**
- **Interrupt and examine a program that is already in progress.**
- **Trace processing of a one-task program by line, instruction, routine, or variable.**
- **Call programs or diagnostic routines directly from the debug program.**
- **Invoke shell commands from debug session.**
- **Examine loaded address maps for a process.**

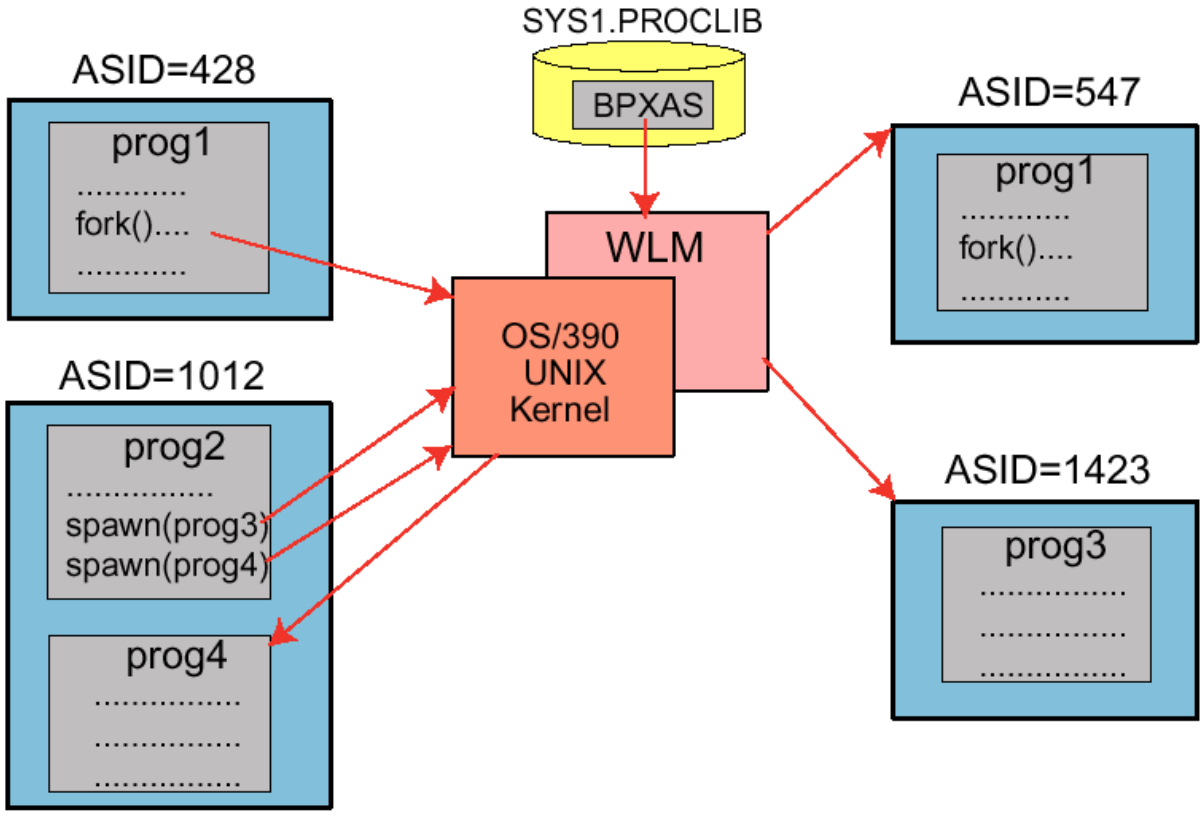



**Copying Source Files from an MVS Data Set to the HFS fromthe Shell**




# A process tree

- A created two child processes, B and C
- B created three child processes, D, E, and F




# z/OS UNIX Prozesse(1)

**Die dem Unix-Prozeß am nächsten kommende MVS-Einheit ist der Adreßraum. Konsequenterweise werden daher Unix-Prozesse, repräsentiert durch eine numerische Prozes-ID, in z/OS-Adreßräumen dargestellt. Der Basisvorgang zum Erzeugen eines neuen Prozesses wird im Unix durch den fork()-Aufruf, ein C/C++ - "Macro" eingeleitet. fork() erzeugt zunächst eine exakte Kopie des laufenden Adreßraums. Abhängig vom Return-Code des fork()-Aufrufes wird**

- **im erzeugenden Adreßraum (Parent) die erhaltene Prozeß-Id verwaltet und mit der (Server-) Funktion fortgesetzt,**
- **im erzeugten Adreßraum (Child) wird dagegen der Parent-(Server-) Code per exec() annulliert und mit der Client-Funktion begonnen.**

# z/OS UNIX Prozesse(2)

**Die fork()-Aufrufe werden durch spezielle Initiatoren, WLM-verwaltete BPXAS-Prozeduren, abgewickelt. Wegen des Kopierens von kompletten Adreßräumen erwies sich der fork()-Vorgang als sehr aufwendig. Es entwickelten sich drei Ansätze zur Lösung dieses Problems:**

1. **Der bereits erwähnte Fork-Initiator unter WLM-Kontrolle ist ein “preiswerter” Weg zum Neuaufbau eines Adreßraums.**
2. **Durch die Implementierung von spawn(), letztlich eine Kombination von fork() und exec() entstand kein unnötiger Kopieraufwand mehr.**
3. **Unix-Prozesse können als z/OS-Subtask im gleichen (erzeugenden) Adreßraum laufen.**

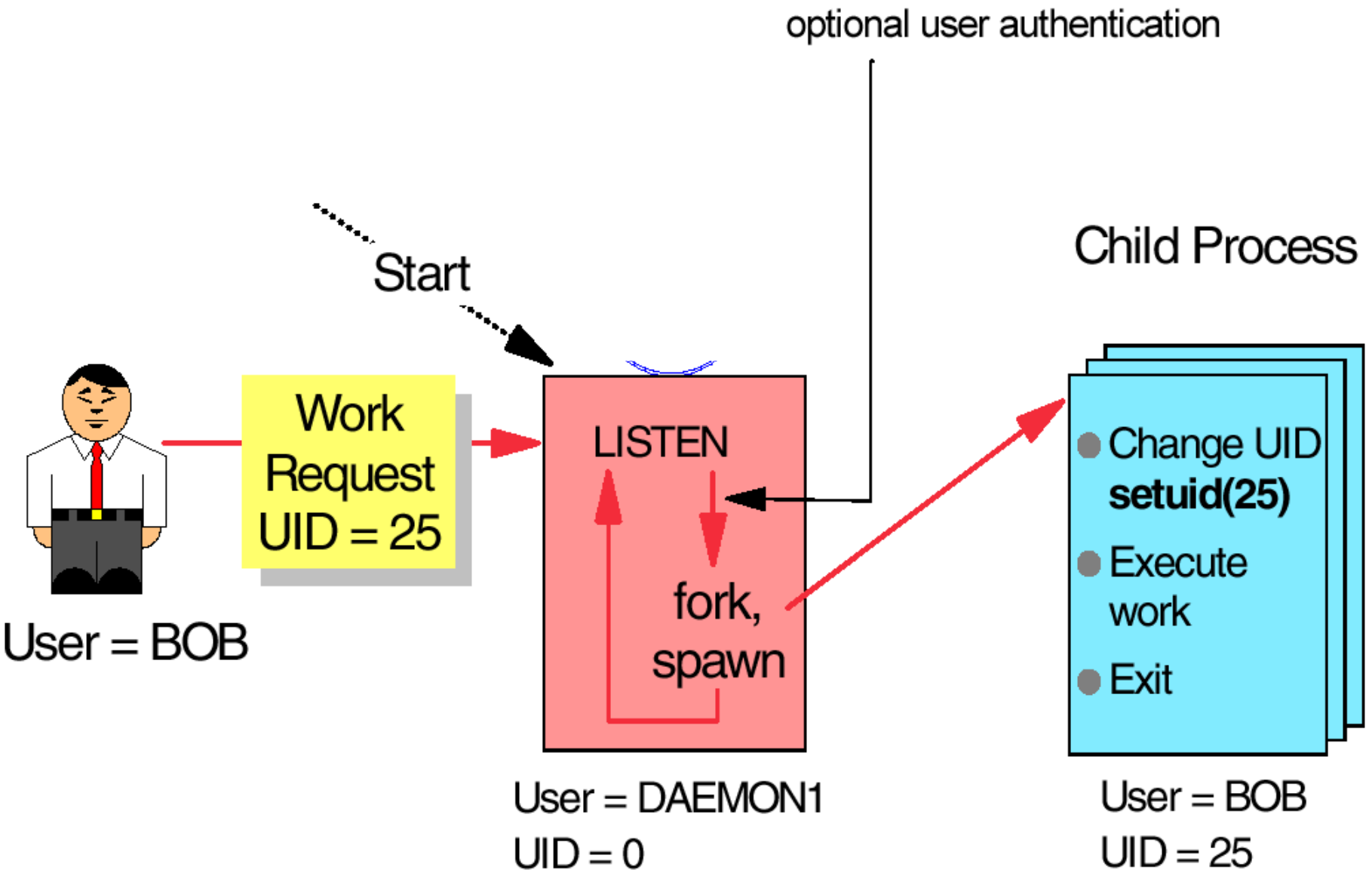
# Daemon Prozesse

**Daemon processes perform continuous or periodic system-wide functions, such as a Web server. Daemons are programs that are typically started when the operating system is initialized and remain active to perform standard services. In z/OS UNIX, daemons can be started as started tasks (STC). Examples of typical UNIX daemons are:**

- **cron, which starts applications at specific times.**
- **inetd, which provides service management for a network.**
- **rlogind, which starts a user shell session when requested to do so by the rlogin command.**

**The UNIX process is an entity that executes a given piece of code, owns the required resources, and is identified using a Process ID (PID).**

**Daemon processes are programs executed in the background, with superuser authority, waiting for services requests. Typically when a request is received and entitled to be served, the daemon duplicates itself using a fork() or spawn() function call,**


optional user authentication
Start
Child Process
Work
Request
UID = 25
LISTEN
fork,
spawn
Change UID
setuid(25)
Execute
work
Exit
User = BOB
User = DAEMON1
UID = 0
User = BOB
UID = 25

# UNIX thread

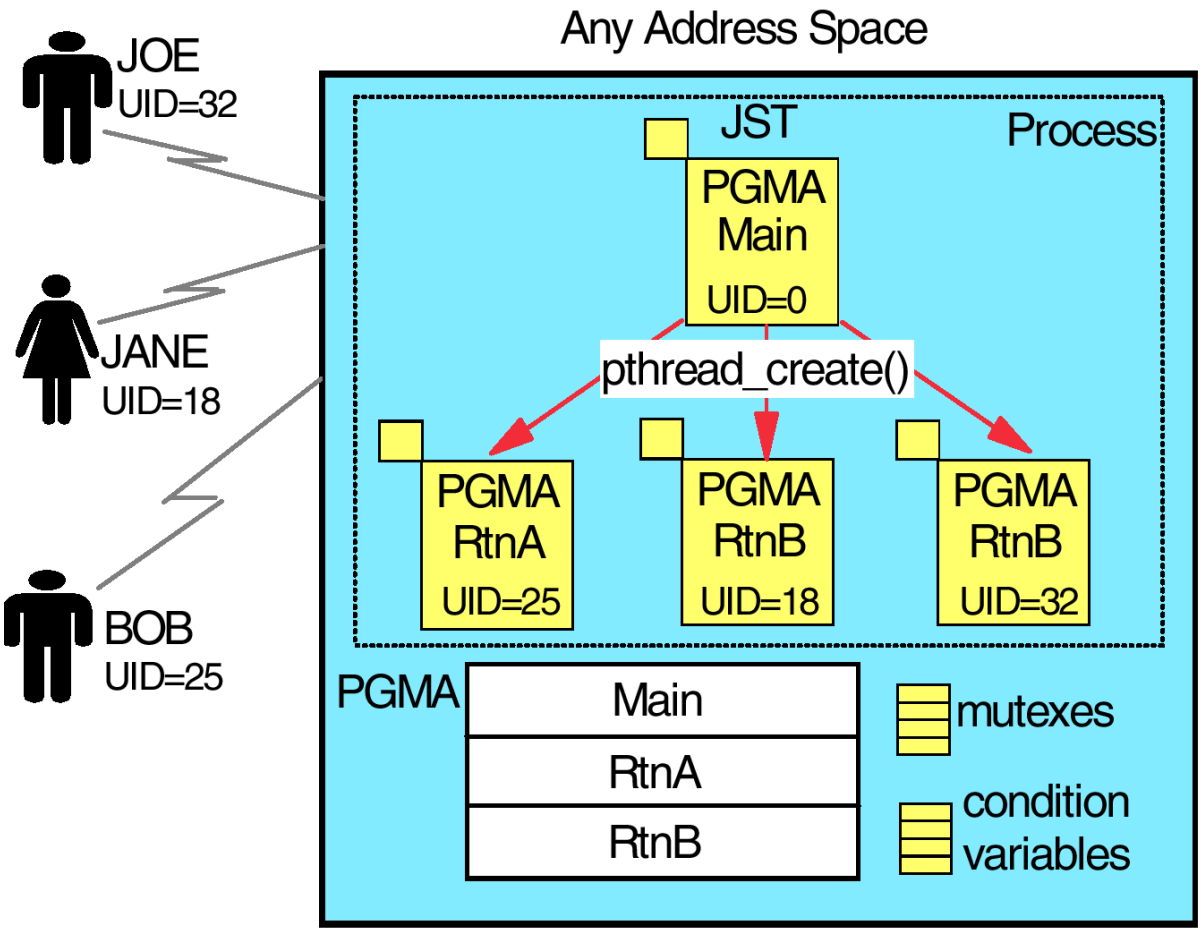
**A server, as opposed to a daemon, can also be defined as a UNIX application servicing client requests, but which uses threading to multitask concurrently executing client requests in the same address space.**

**When a client request is received, the server main process uses the pthread_create() call to create a new thread, which executes the client request asynchronously.**

**Thread support improves performance by providing concurrent, asynchronous processing without the requirement to create a new address space and a new process to run the client request.**

**In the z/OS UNIX implementation, this translates into the main process being the main MVS task in the address space and the threads being started as MVS subtasks. Each thread has a dedicated TCB, and can execute a different code subroutine**

**Each thread (task) can be initialized to run in different security environments.**


Any Address Space
JOE
UID=32
JANE
UID=18
BOB
UID=25
JST
Process
PGMA
Main
UID=0
pthread_create()
PGMA
RtnA
UID=25
PGMA
RtnB
UID=18
PGMA
RtnB
UID=32
PGMA
Main
RtnA
RtnB
mutexes
condition
variables

# UNIX thread

# Process Management

An OS/390 UNIX System Services process maps into an OS/390 address space and an OS/390 task environment defined by a task control block (TCB) and related control blocks that exist for the process. In addition to the TCB, the OS/390 UNIX System Services kernel address space contains a number of control blocks that represent an OS/390 UNIX System Services process. There are situations where multiple processes may exist within the same OS/390 address space, and in such cases a process may be running as a job step task or subtask.

A started task in OS/390 is similar to a UNIX daemon, a long-running process. A process maps to an MVS address space and an MVS task environment exists for the process, in terms of a task control block (TCB) and related control blocks. In addition to the TCB, the OE Kernel address space maintains
a number of control blocks that represent an OpenEdition process. These control blocks exist within the OE Kernel address space and are created whenever an existing standard MVS program begins using OpenEdition services (the MVS address space is being dubbed as an OpenEdition process) or a new OpenEdition process is being created by the OpenEdition process creation functions. By default, a new process will run as the job step task in an MVS address space. There are situations where more
processes may exist within the same MVS address space, and in such a case a process may be running as a subtask of the job step task.

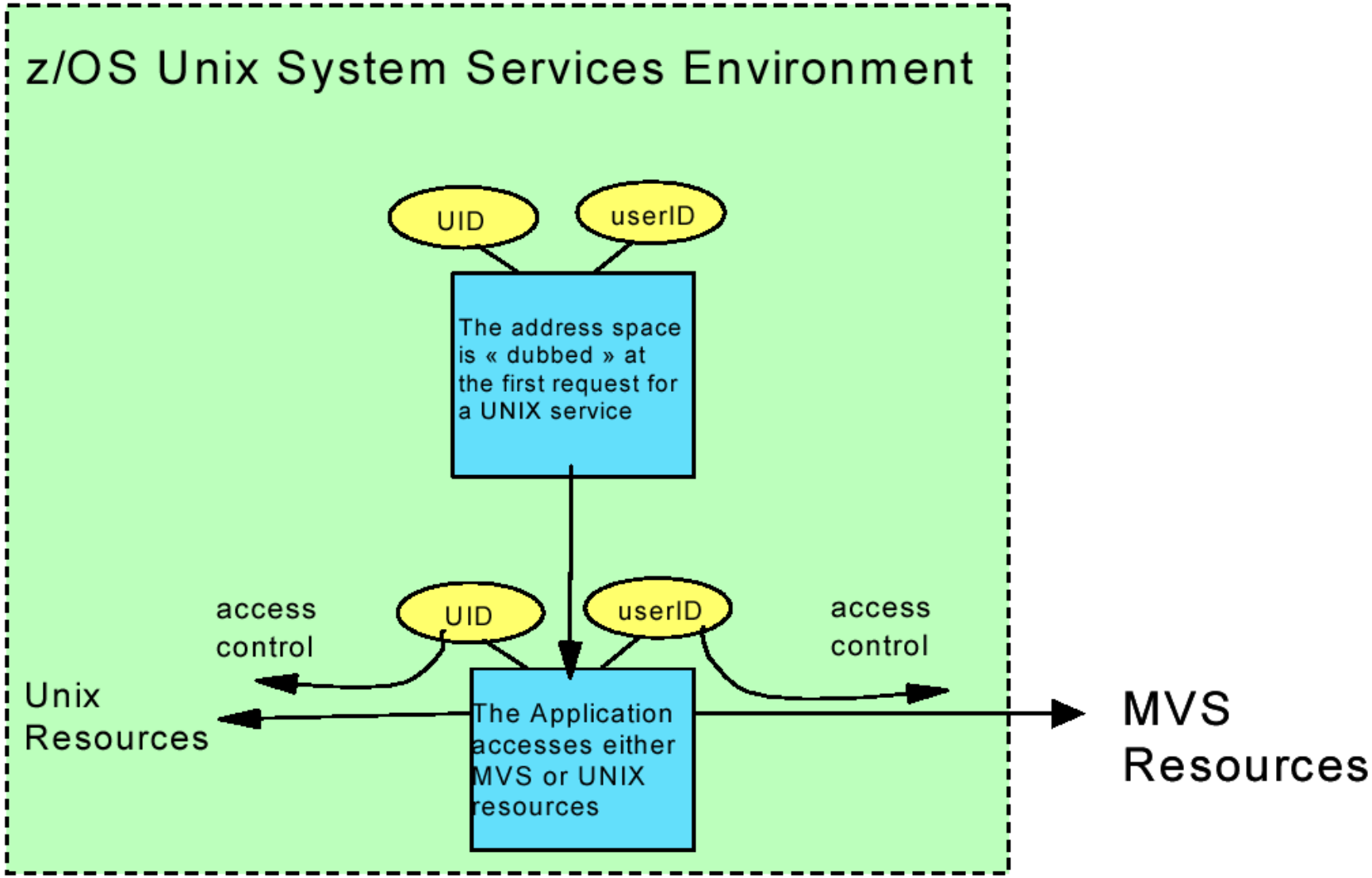
To start a new process, a process may use the fork() service as in any UNIX environment. In OS/390 this service invokes a Workload Manager (WLM) controlled initiator. The initiator address space name is BPXAS. BPXAS is a special initiator with special interfaces to WLM and UNIX
System Services.

Fork and spawn use a WLM service to get an address space to process a request. The WLM service checks its queue to see if there is an idle address space from a free pool of BPXAS address spaces that are waiting for work. If there are no idle BPXAS address spaces, WLM determines whether a new one can be created based on system load. If the free pool is empty, an ASCRE is done using the name BPXAS as the procedure to start a new one. When a BPXAS address space is finished, the address space goes back into the free pool. After 30 minutes of being idle, the address space is terminated.

The spawn service is another way to create a new process. It looks like fork(), except that the new process is not a copy of the parent process. The child process inherits file and socket descriptors, as with the fork(). With a spawn, the child process can be created in the parent address space.
OS/390 UNIX System Services dub the OS/390 address space as a process. In order to do that, new information is added to the current address space:

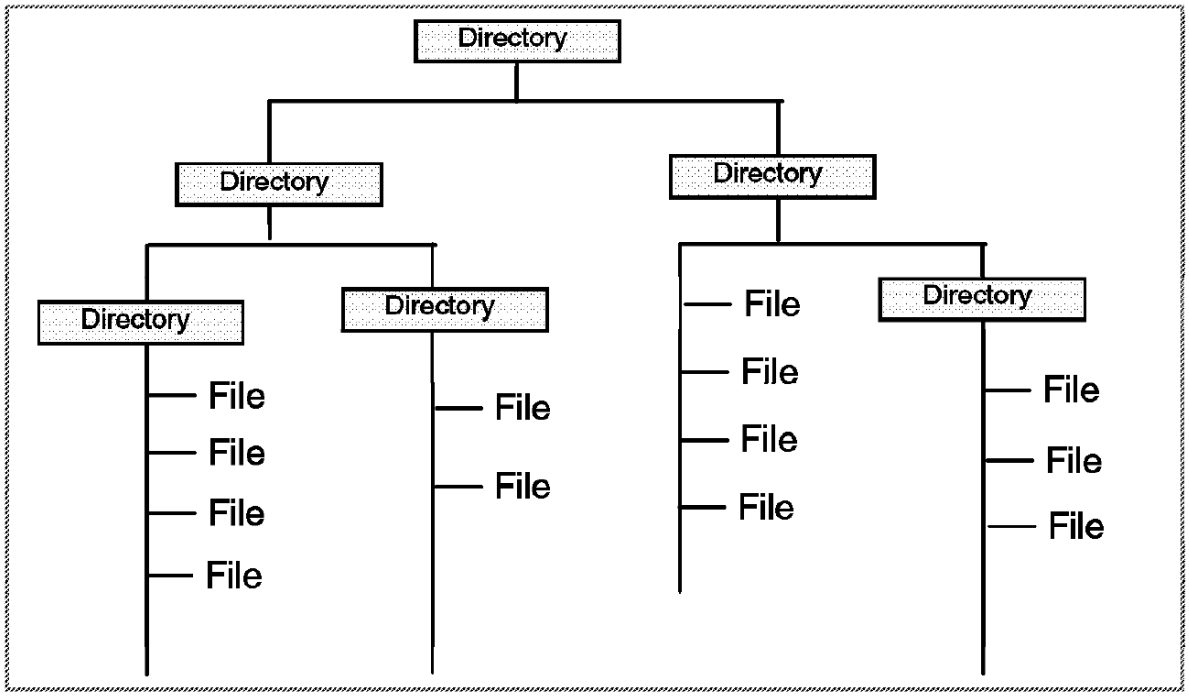
- Name Definition
- Real UID Identifies the user who created the process
- Effective UID Determines the owner access privileges of a process
- Real GID Identifies the current connect group of the user for which the process was created
- Effective GID Determines the group access privileges of a process

The real UID and GID indicate who the user is, and the effective UID and GID are used for file access permission.


z/OS Unix System Services Environment
UID
userID
The address space is « dubbed » at the first request for a UNIX service
access control
UID
userID
access control
Unix Resources
The Application accesses either MVS or UNIX resources
MVS Resources

# Authentification

**z/OS UNIX users are authenticated by RACF through the System Authorization Facility (SAF) interface, by using their MVS user ID (which equates their UNIX user name) and their MVS password,**



HFS Data Set

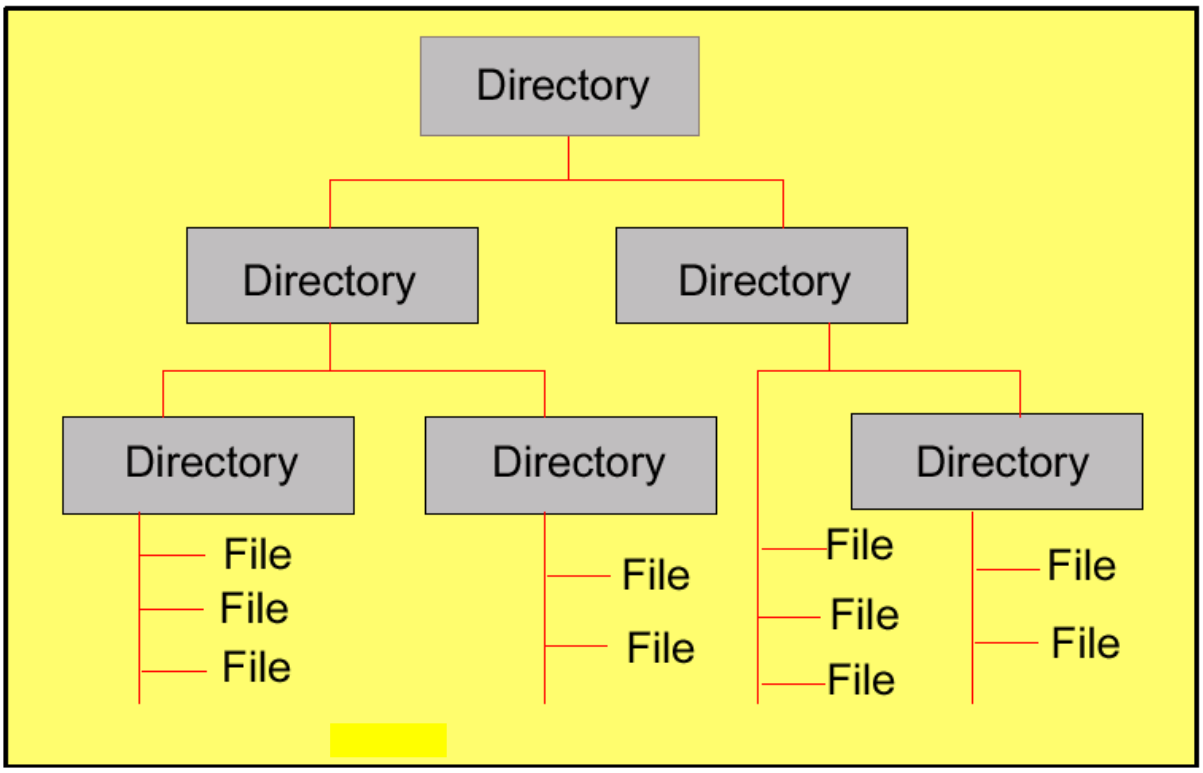
# Hierarchical File System

**The XPG4 file system is hierarchical and byte oriented. Finding a file in this file system is done by searching a directory or a series of directories. There is no concept of an z/OS catalog that points directly to a file.**

**A path name identifies a file and consists of directory names and a file name. A fully-qualified file name which consists of the name of each directory in the path to a file plus the file name itself, can be up to 1023 bytes long.**

**The hierarchical file system allows for file names in mixed case.**

**The Hierarchical File System (HFS) data set which contains the hierarchical file system is a new z/OS data set type. An HFS data set is allocated as any other z/OS data set and it must be managed by DFSMS.**




# HFS Data Set

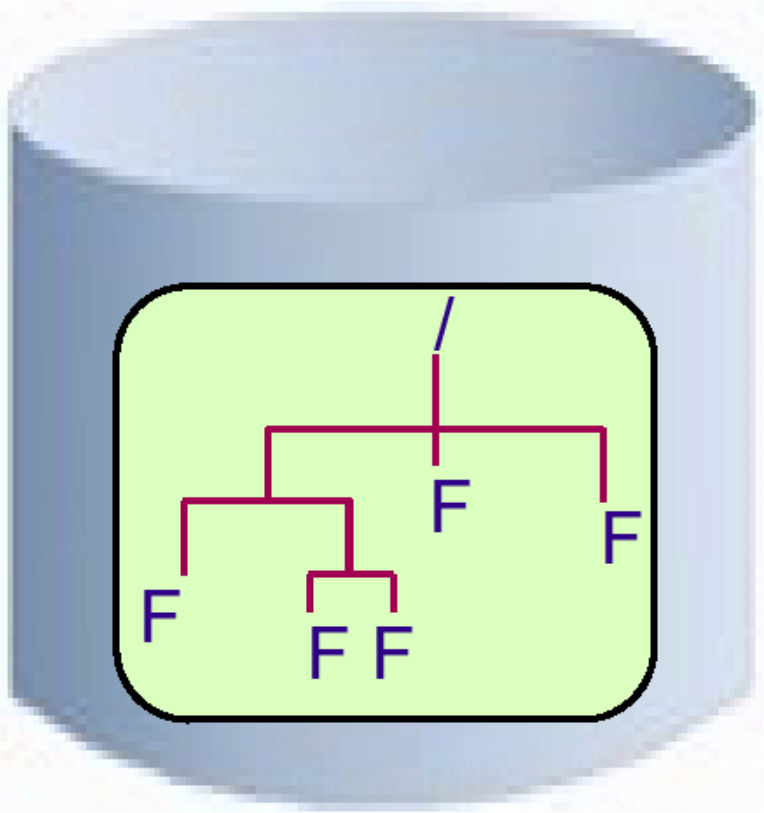
**Unix-Dateisysteme sind u.a. durch folgende Merkmale geprägt:**

- **Alle Dateien sind Bestandteil eines Verzeichnisbaumes, dessen Wurzel (Root) das Directory “/” ist.**
- **Der Gesamt-Verzeichnisbaum kann sich aus mehreren Teilbäumen durch Mount-Vorgänge zusammensetzen.**
- **Dateien haben in der Regel keine dem Betriebssystem bekannte Record-Struktur. Dieses bleibt jeweils den Anwendungen überlassen.**

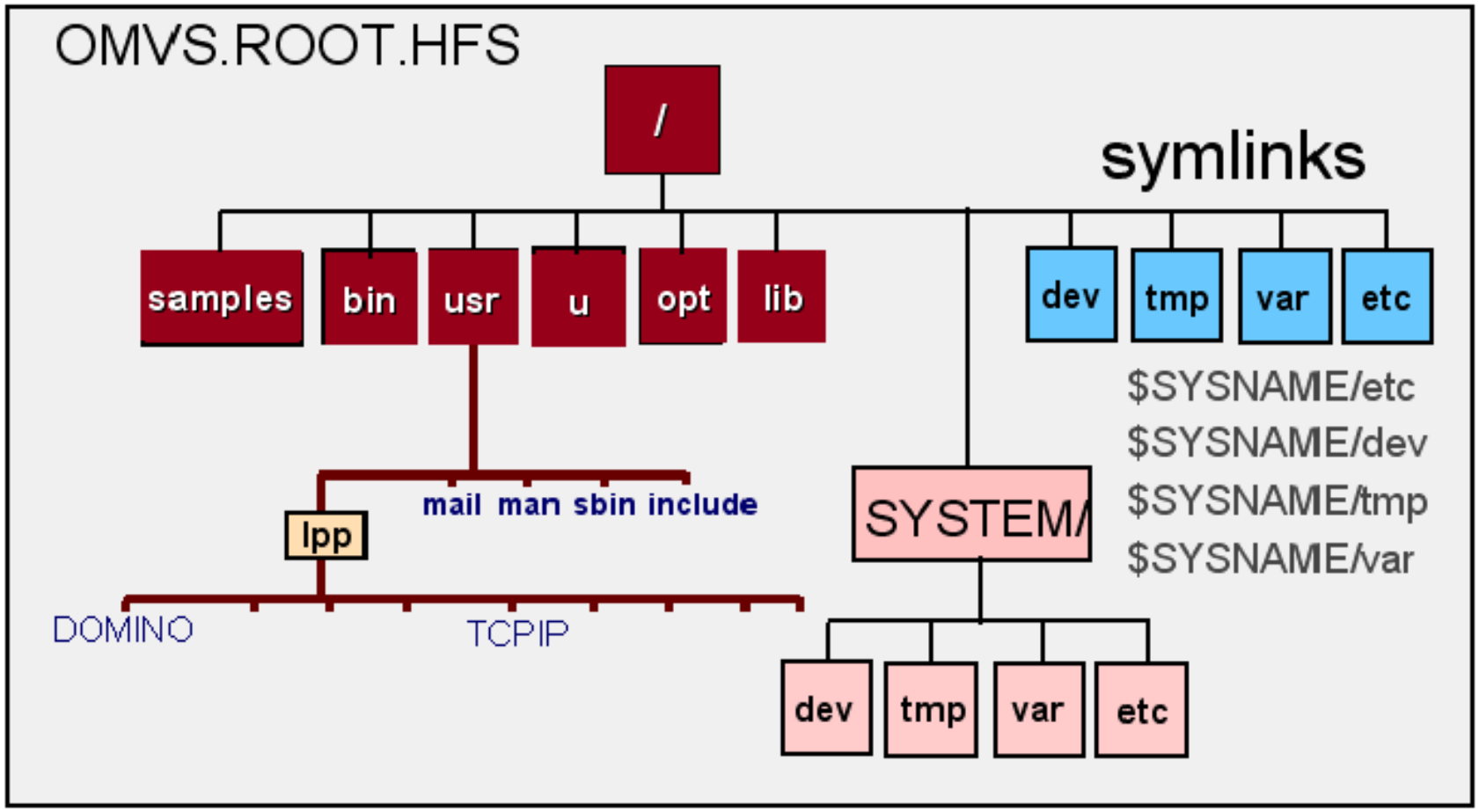
**Das z/OS Unix hat für diese Art der Datenhaltung einen speziellen Dateityp entwickelt, das HFS-Dataset.Herkömmlichen z/OS Utility-Programmen bleibt die interne Struktur dieser File-Systeme verborgen, obwohl eine gewisse Ähnlichkeit mit PDSEs besteht.**

**Individuelle Unix-Prozesse müssen die Kernel-Dienste in Anspruch nehmen, um Daten zu lesen und zu schreiben. Wie im Unix üblich sieht ein Programm nur das sogenannte logische Filesystem. Ein HFS-Dataset stellt ein physikalisches Filesystem dar.**

# HFS Data Sets

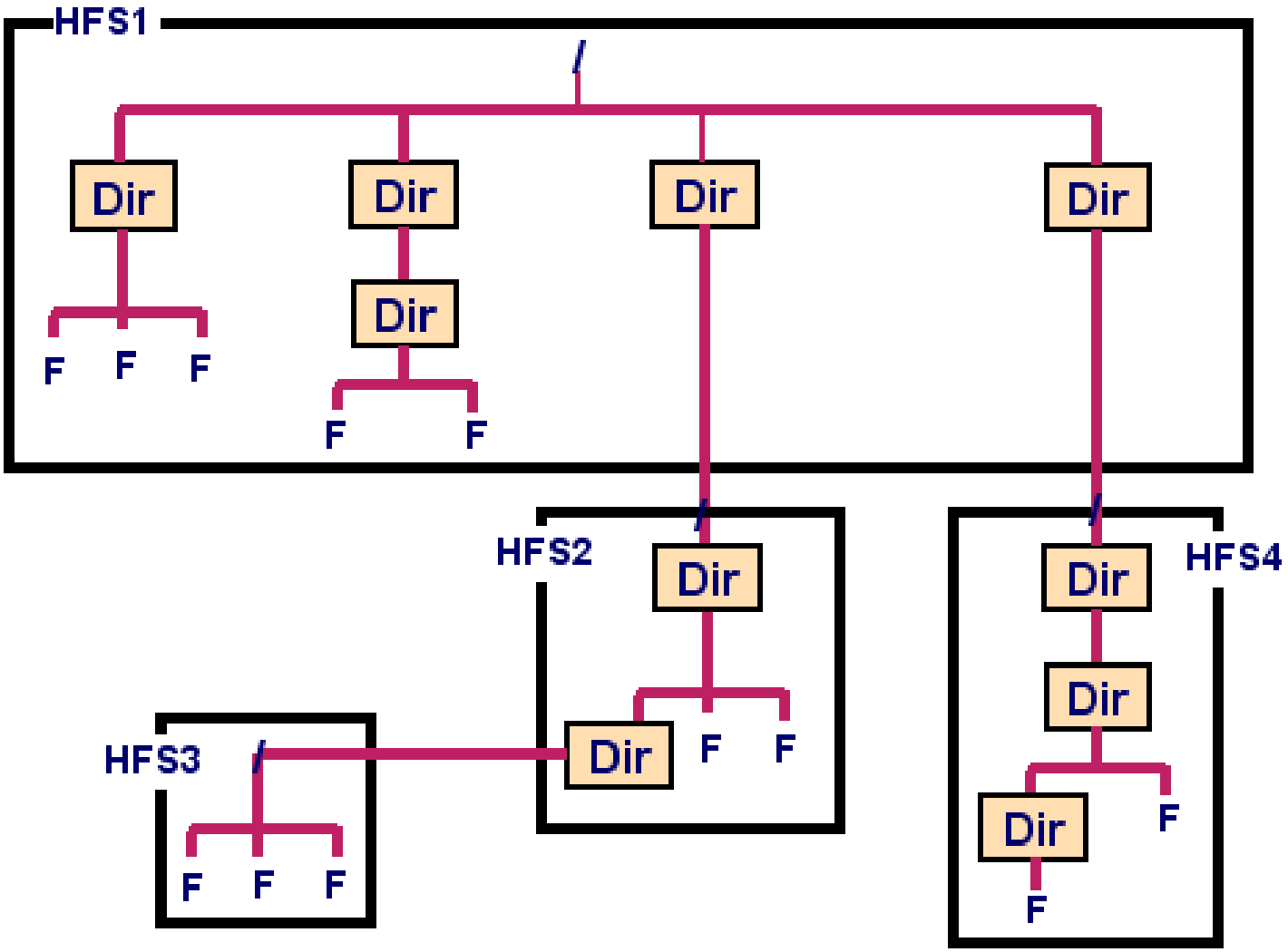


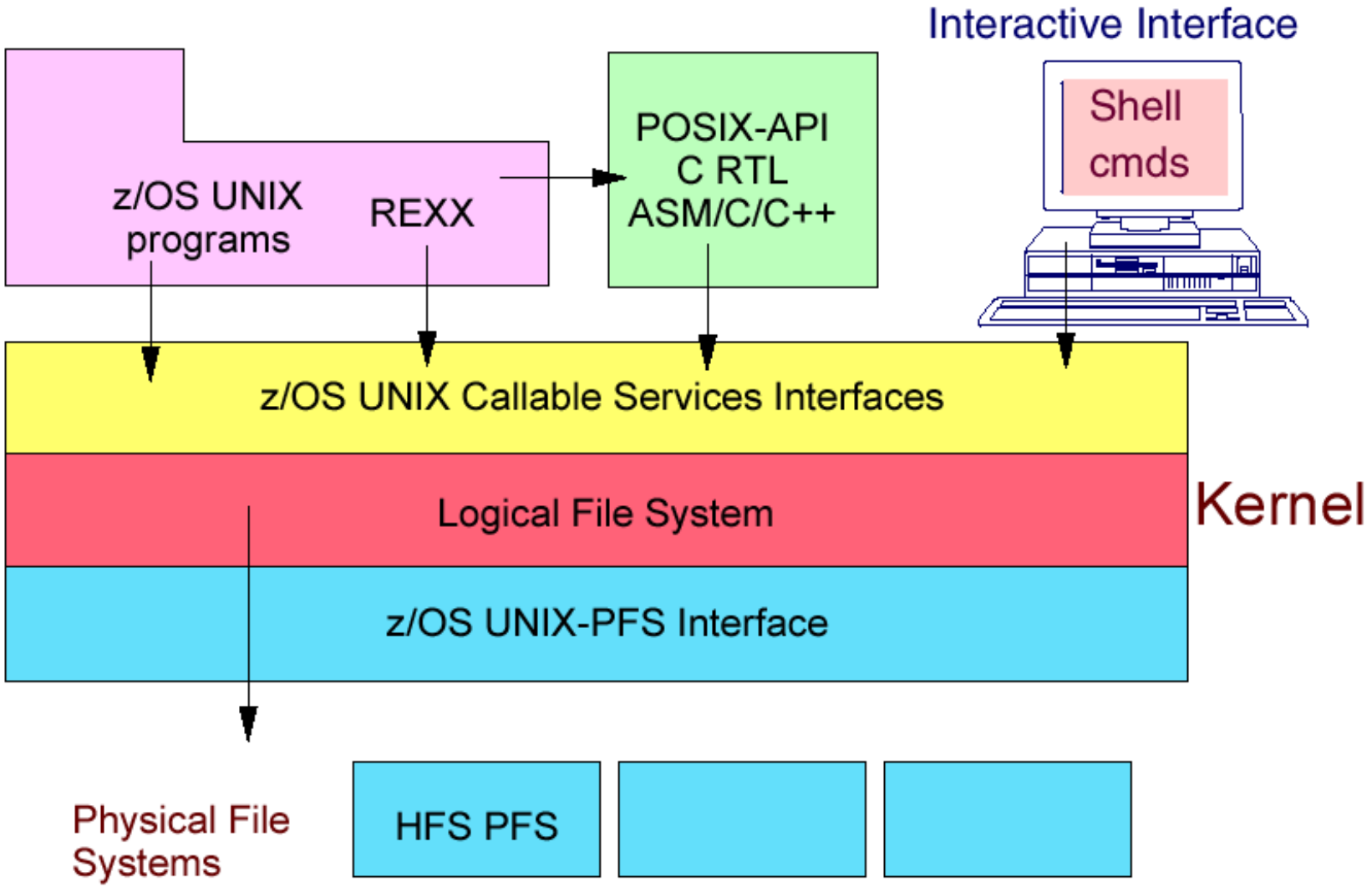

- SMS-Managed
- DSNTYPE=HFS
- Max 123 Extents
- Single Volume
- Max File Size = 1 physical volume (2.8 GB) - 3390-3




**Root File System**

# Multiple File Systems




Interactive Interface
Shell
cmds
z/OS UNIX
programs
REXX
POSIX-API
C RTL
ASM/C/C++
z/OS UNIX Callable Services Interfaces
Logical File System
Kernel
z/OS UNIX-PFS Interface
Physical File
Systems
HFS PFS

# z/OS UNIX file system

**Applications in z/OS UNIX get access to data through physical file systems (PFSs), which can be thought of as a set of access method services with proper APIs to write and read data. The underlying physical file systems are defined in the BPXPRMxx member of SYS1.PARMLIB (with the FILESYSTYPE statement) in order to be automatically activated when z/OS UNIX is started.**

**The following are all PFSs:**

## Hierarchical File System (HFS)

**z/OS UNIX files are organized in a HFS, as in other UNIX systems. Files are members of a directory, and each directory is in turn a member of another directory at a higher level, the highest level being the *root* directory. The HFS is implemented in z/OS using HFS or zFS data sets.**

## Network File System (NFS)

**The NFS client on z/OS UNIX can mount a file system, directory, or file from any system in the network that runs an NFS server, within a z/OS UNIX user's directory. The remote files and directories can then be worked on as though they were local z/OS UNIX files and directories.**

## Distributed File System (DFS™)

**DFS joins the local file systems of several file server machines making the files equally available to all DFS client machines. DFS allows users to access and share files stored on a file server anywhere in the network, without having to consider the physical location of the file.**

## Temporary File System (TFS)

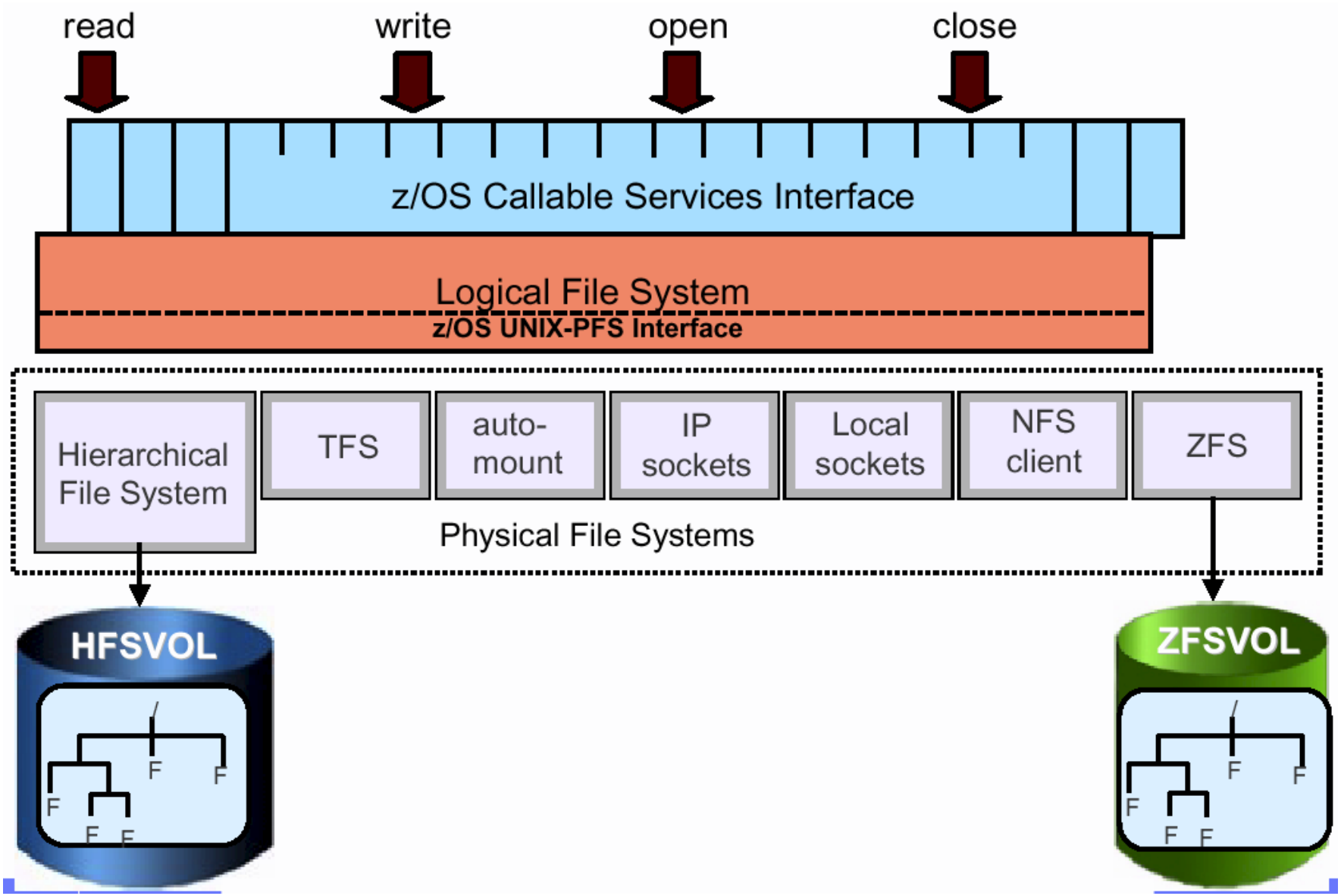
**The TFS is an in-memory physical file system that delivers high-speed access to data. A TFS is usually mounted at the /tmp directory, therefore it can be used as a high-speed file system for temporary files.**

## Pipe

**A program creates a pipe with the pipe() function. A pipe typically is written with data by one process to be read by another process. The two ends of a pipe can also be used in a single program task. A pipe does not have a name in the file system, and it vanishes when the last process using it closes it.**

## Socket

**A program creates a socket with the socket() function. A socket is a method of communication between two processes that allows communication in two directions, in contrast to pipes, which allow communication in only one direction. The processes using a socket can be on the same system or on different systems in the same network. A PFS is specialized for a type of file system, with file system having a quite broad meaning in UNIX as it can designate data residing on a disk as well as data being transmitted through sockets.**


read
write
open
close
z/OS Callable Services Interface
Logical File System
z/OS UNIX-PFS Interface
Hierarchical File System
TFS
auto-mount
IP sockets
Local sockets
NFS client
ZFS
Physical File Systems
HFSVOL
ZFSVOL

# zFS

**zFS is a new UNIX file system for z/OS**

**A component of the Distributed File System (DFS) since 1995**

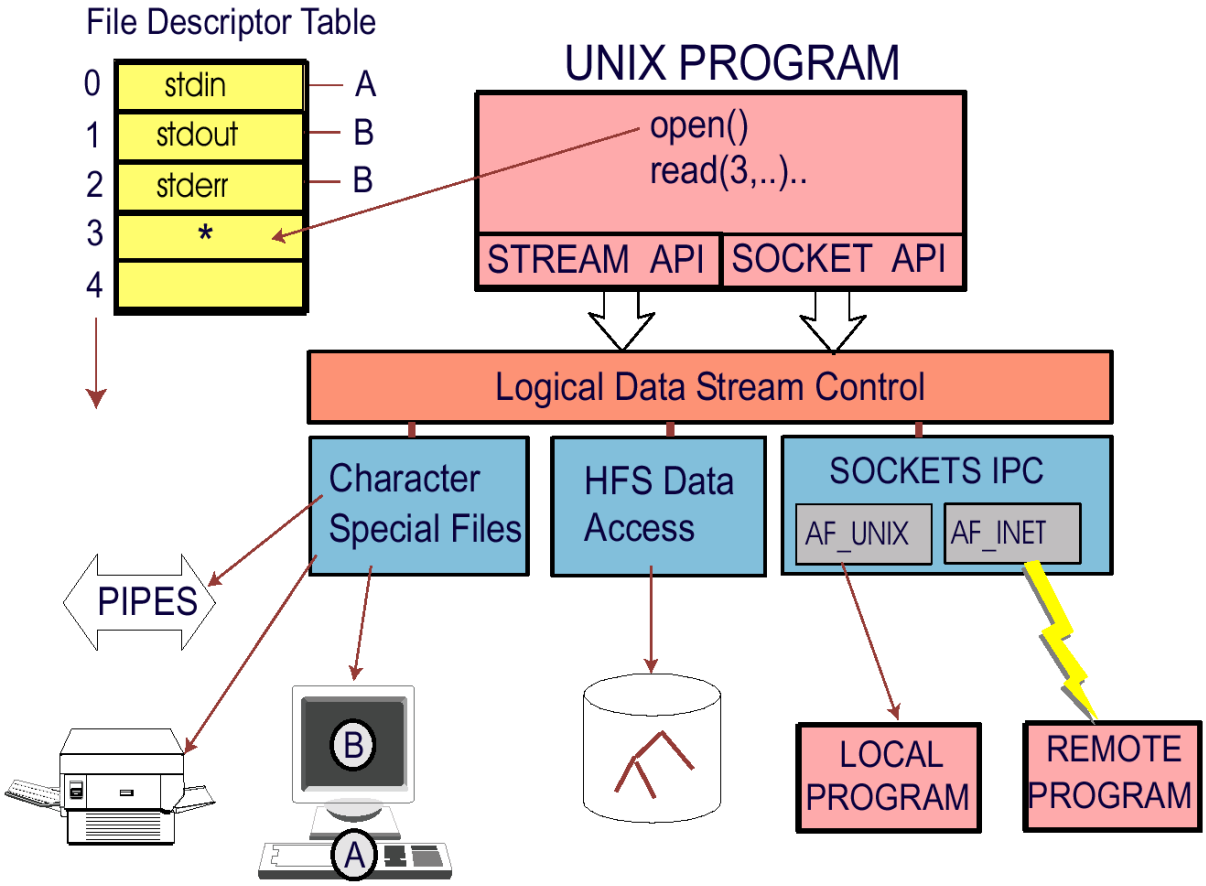
**Does not replace HFS**

**Using zFS, you can:**

- **Run applications, just like HFS**
- **Replace HFS or use both file systems together**

**Advantages of zFS**

- **Better performance**
- **Enhanced administrative functions**
- **Same interface to applications as HFS**
- **Improved crash recovery**



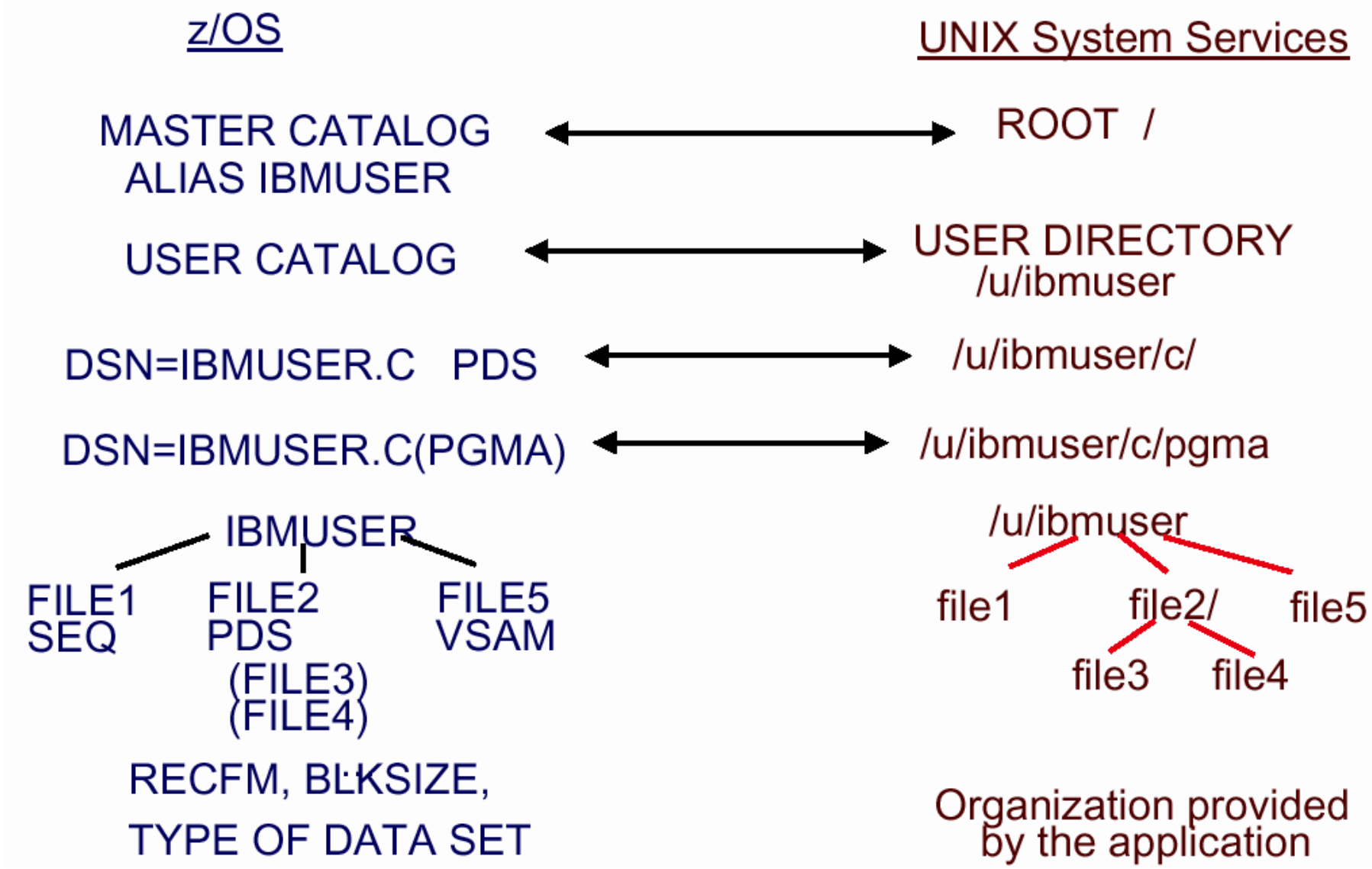

# Datenzugriff im Unix

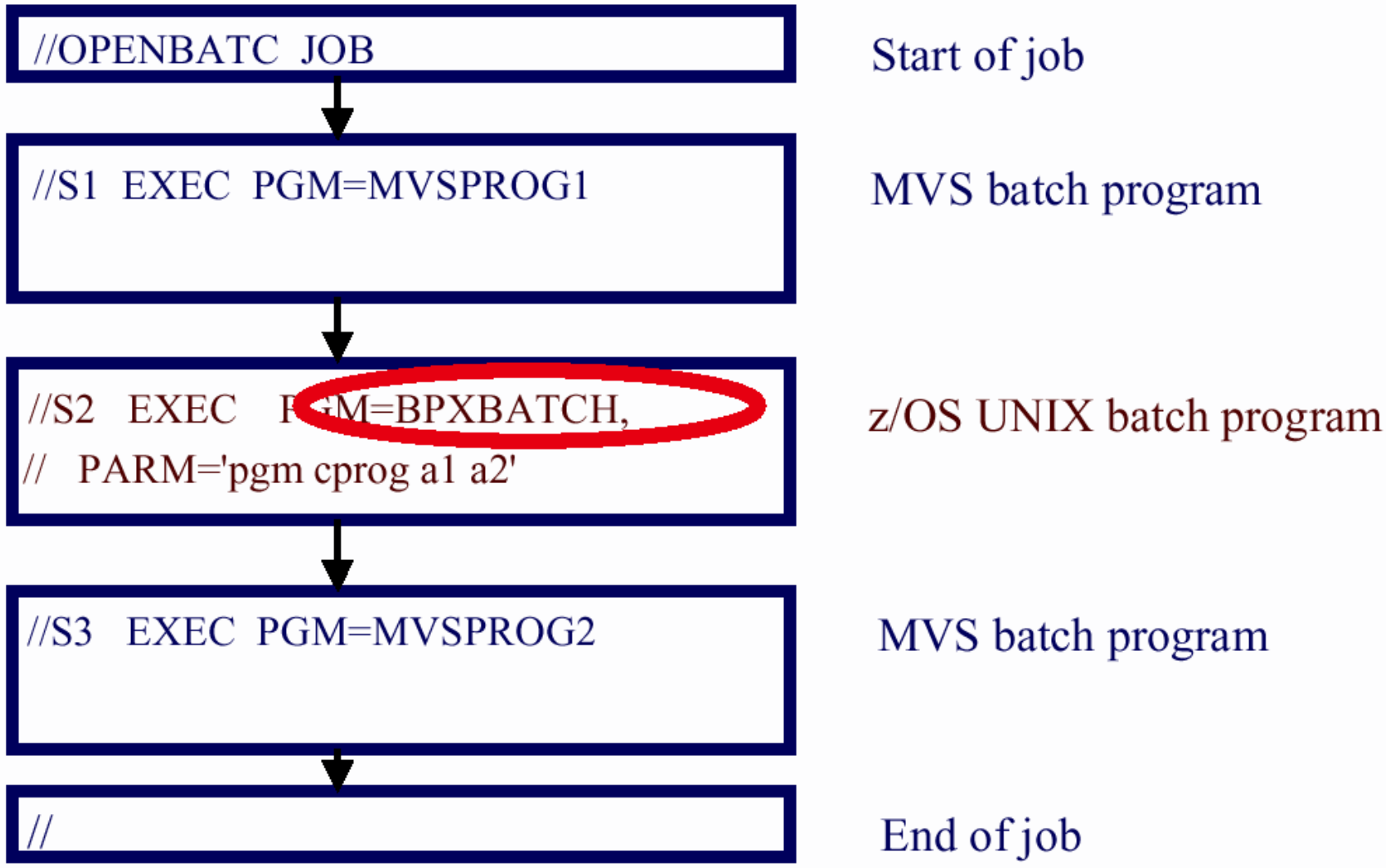
**Unix-Programme, repräsentiert durch Prozesse, werden von der physikalischen Beschaffenheit einer Datei durch das logische und die physikalischen Dateisysteme des Kernels abgeschirmt. Sie erhalten nach einem open()-Aufruf lediglich einen File Descriptor (ein Index in eine Prozess-weise File-Descriptor Tabelle) zurück.**

**Dieser Ansatz gestattet eine bemerkenswerte Portabilität bei Anwendungsprogrammen: mit relativ geringen Anpassungen kann ein Programm anstatt auf Dateien zuzugreifen mit anderen Programmen sei es local oder remote kommunizieren.**

**Wie im Unix üblich sieht ein Programm nur das sogenannte logische Filesystem. Ein HFS-Dataset stellt aber ein physikalisches Filesystem dar.**

**Neben HFS-Dateien sind TCPIP-Socket-Verbindungen und Unix-interne Programm-zu-Programm-Verbindungen weitere physikalische Filesysteme.**


z/OS
UNIX System Services
MASTER CATALOG
ALIAS IBMUSER
ROOT /
USER CATALOG
USER DIRECTORY
/u/ibmuser
DSN=IBMUSER.C PDS
/u/ibmuser/c/
DSN=IBMUSER.C(PGMA)
/u/ibmuser/c/pgma
IBMUSER
FILE1
SEQ
FILE2
PDS
(FILE3)
(FILE4)
FILE5
VSAM
/u/ibmuser
file1
file2/
file5
file3
file4
RECFM, BLKSIZE,
TYPE OF DATA SET
Organization provided
by the application

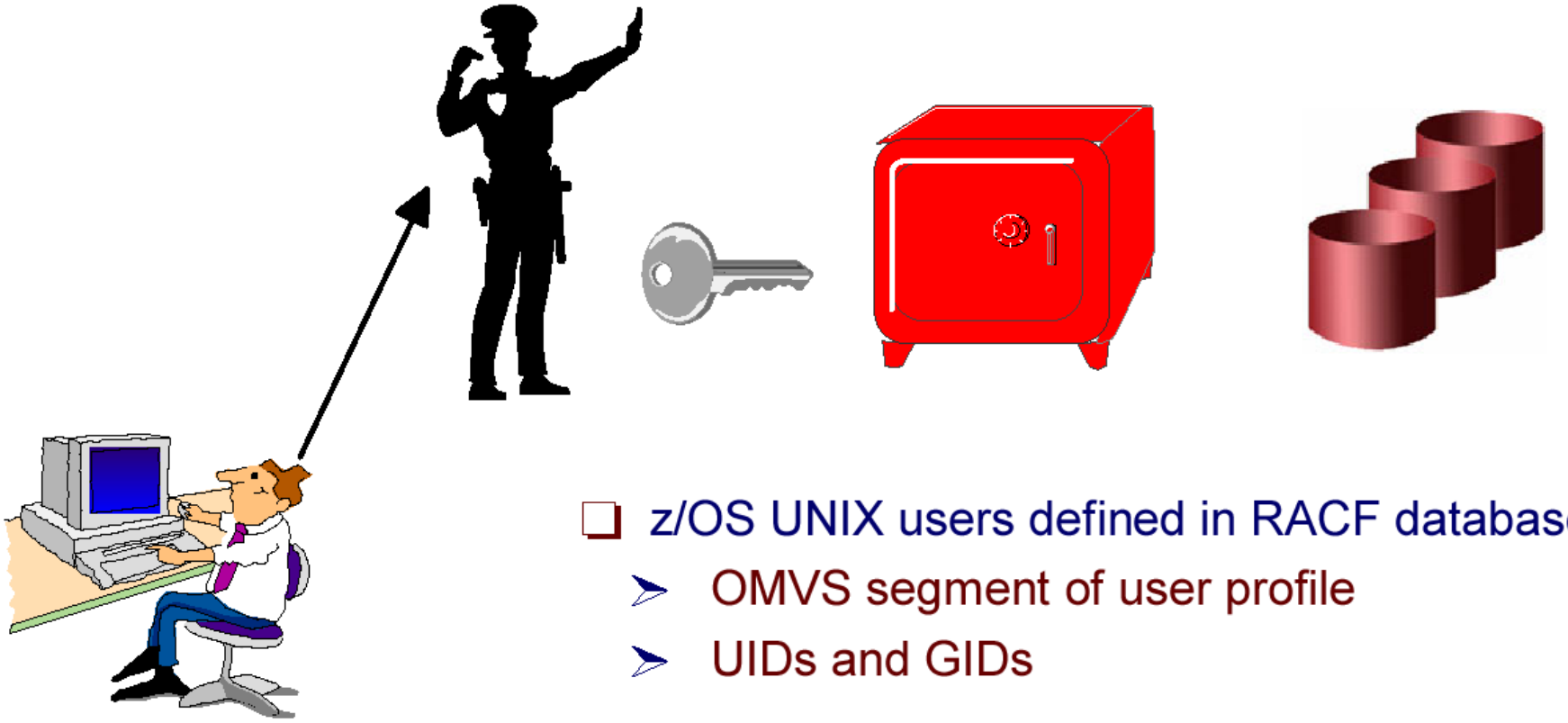

//OPENBATC JOB
Start of job
//S1 EXEC PGM=MVSPROG1
MVS batch program
//S2 EXEC PGM=BPXBATCH,
// PARM='pgm cprog a1 a2'
z/OS UNIX batch program
//S3 EXEC PGM=MVSPROG2
MVS batch program
//
End of job

## Die wichtigste Hilfe:

## Using the man Command

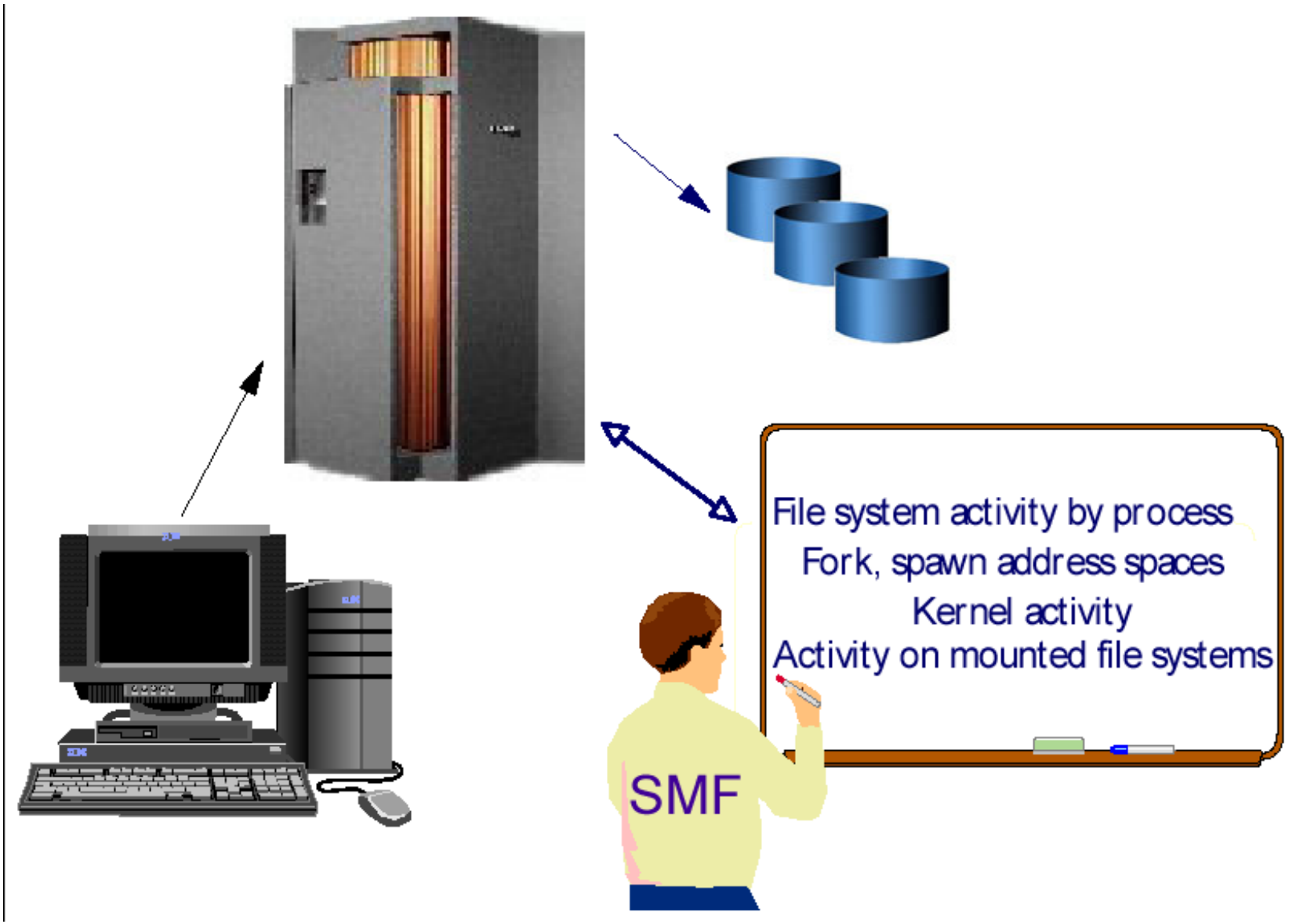
- man command gives help information about a shell command
  - man command_name
- To produce a list of all the shell commands for editing
  - man -k edit
- To view descriptions of TSO/E commands such as mount
  - man tsomount

- z/OS UNIX requires a security product



- z/OS UNIX users defined in RACF database
  - OMVS segment of user profile
  - UIDs and GIDs

| Userid | Default Group | Connect Groups | | TSO | DFP | OMVS | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | UID | Home | Program |
| SMITH | PROG1 | PROG1 | PROG2 | ... | ... | 15 | /u/smith | /bin/sh |


File system activity by process
Fork, spawn address spaces
Kernel activity
Activity on mounted file systems
SMF

# SMF and z/OS UNIX

**System Management Facility (SMF), which is a component of the BCP element, collects data for accounting. SMF job and job-step accounting records identify processes by user, process, group, and session identifiers. Fields in these records also provide information on resources used by the process. SMF file system records describe file system events such as file open, file close, and file system mount, unmount, quiesce, and unquiesce.**

**You can use SMF to report on activity from a user application, to report activity on a job and jobstep basis, and to report activity of mounted file systems and files.**

**SMF job and job-step accounting records identify z/OS UNIX processes by:**

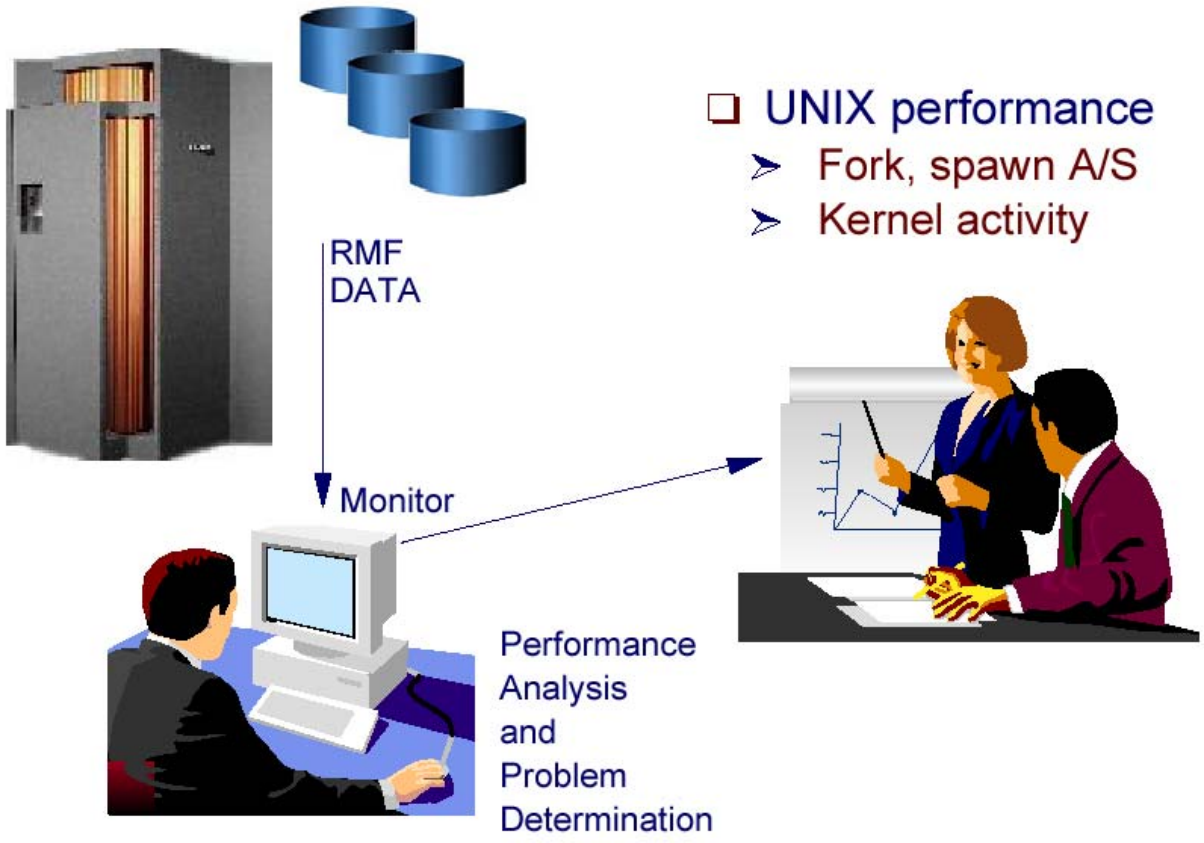
- **User (UID)**
- **Process (z/OS address space)**
- **Group (GID)**
- **Session identifiers**

RMF
DATA

Monitor

Performance
Analysis
and
Problem
Determination

- UNIX performance
  - Fork, spawn A/S
  - Kernel activity




# RMF and z/OS UNIX

# RMF and z/OS UNIX

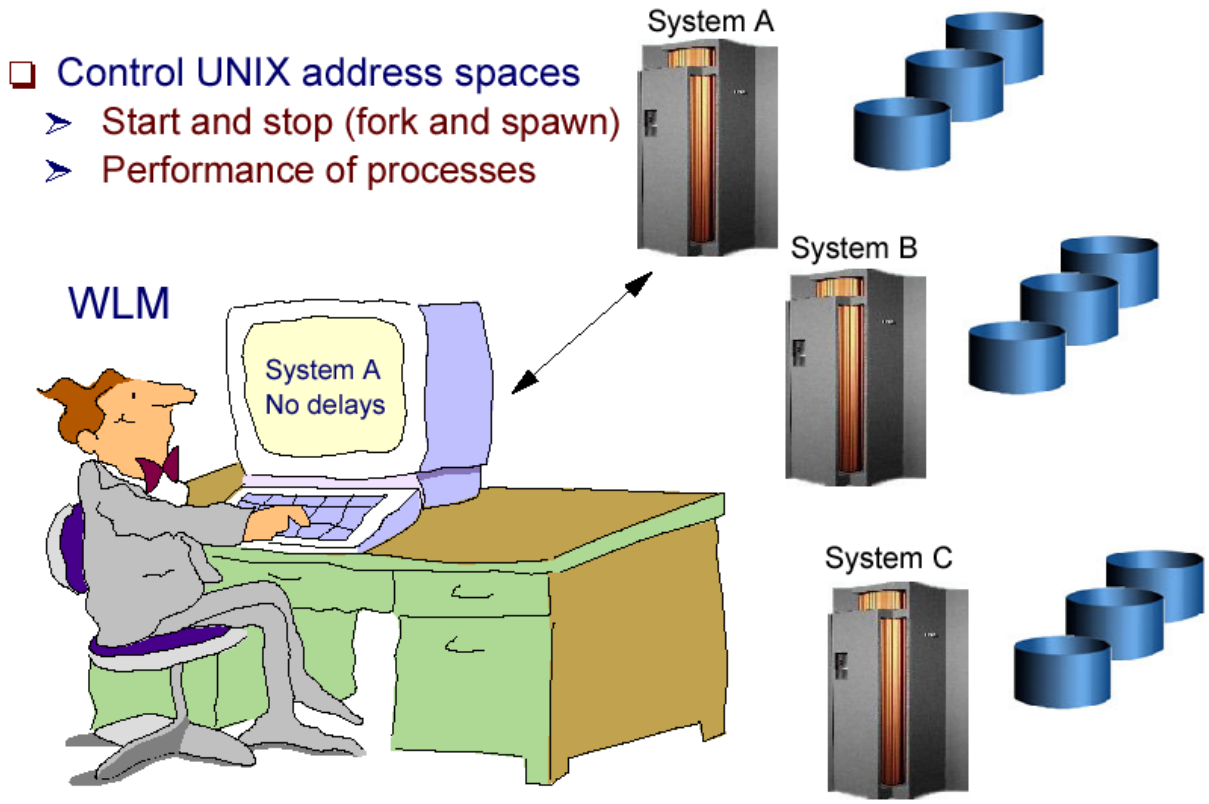
**Resource Measurement Facility (RMF) collects data used to describe z/OS UNIX performance. RMF reports support an address space type of OMVS for address spaces created by fork or spawn callable services and support two swap reason codes.**

**RMF monitors the use of resources in an OMVS Kernel Activity report.**

**The software products supporting system programmers and operators in managing their systems heavily influence the complexity of their jobs, and their ability to keep systems at a high level of availability.**

**RMF collects data used to describe z/OS UNIX performance. RMF reports support an address space type of OMVS for address spaces created by fork or spawn callable services.**

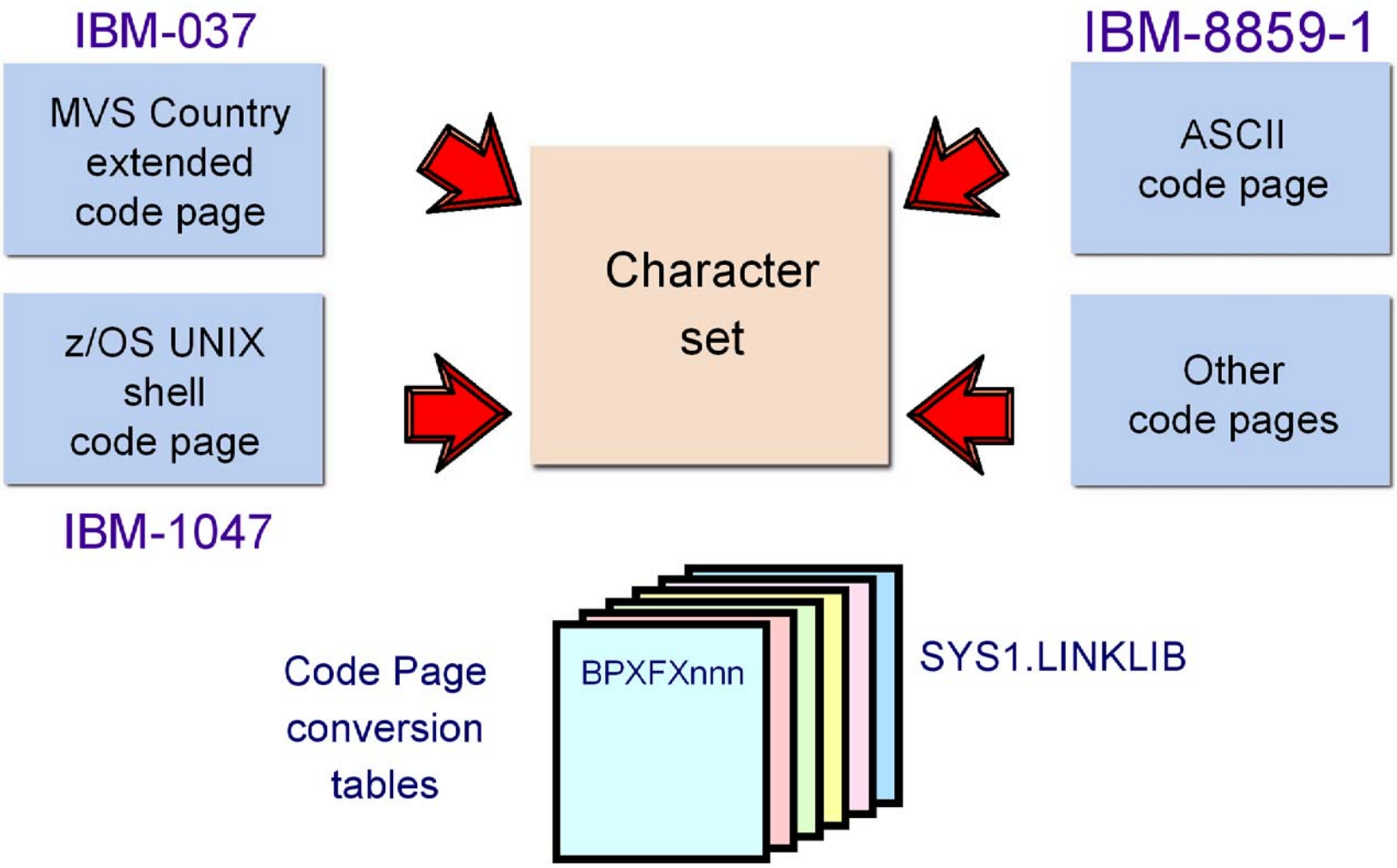
**When an installation specifies an OMVS subsystem type in the workload manager service policy, RMF shows the activity of forked address spaces separately in the RMF Workload Activity report.**




**The purpose of workload management is to balance the available system resources to meet the demands of S/390 subsystem work managers such as CICS, BATCH, TSO, UNIX Services, and WebServer, in response to incoming work requests.**

**The workload manager is a component of the BCP element. When using WLM, you do not need to do any tuning or issue any commands. The kernel uses WLM to create child processes when running in goal mode or compatibility mode, or both.**

**When programs issue fork or spawn, the BPXAS PROC found in SYS1.PROCLIB is used to provide a new address space. For a fork, the system copies one process, called the parent process, into a new process, called the child process. Then it places the child process in a new address space. The forked address space is provided by WLM.**


IBM-037
MVS Country extended code page
z/OS UNIX shell code page
IBM-1047
Character set
IBM-8859-1
ASCII code page
Other code pages
Code Page conversion tables
BPXFXnnn
SYS1.LINKLIB

# Code Page, Locale

**Ein „Character Set“ ist eine Menge von alpha-numerischen Zeichen**

**Eine „Code Page“ ist eine Zuordnung von hexadezimalen Werten zu den Zeichen eines Charakter Sets.**

**Eine „Locale“ beschreibt eine kulturelle Umgebung. Hierzu gehören neben einer Code Page Spezifikation:**

- **Sortierfolge**
- **Datum Konventionen**
- **Uhrzeit Konventionen**
- **Darstellung numerischer Daten (z.B. Dezimal Komma, Dezimalpunkt)**
- **Darstellung von Geldbeträgen**

**Wichtige OS/390 Code Pages sind:**

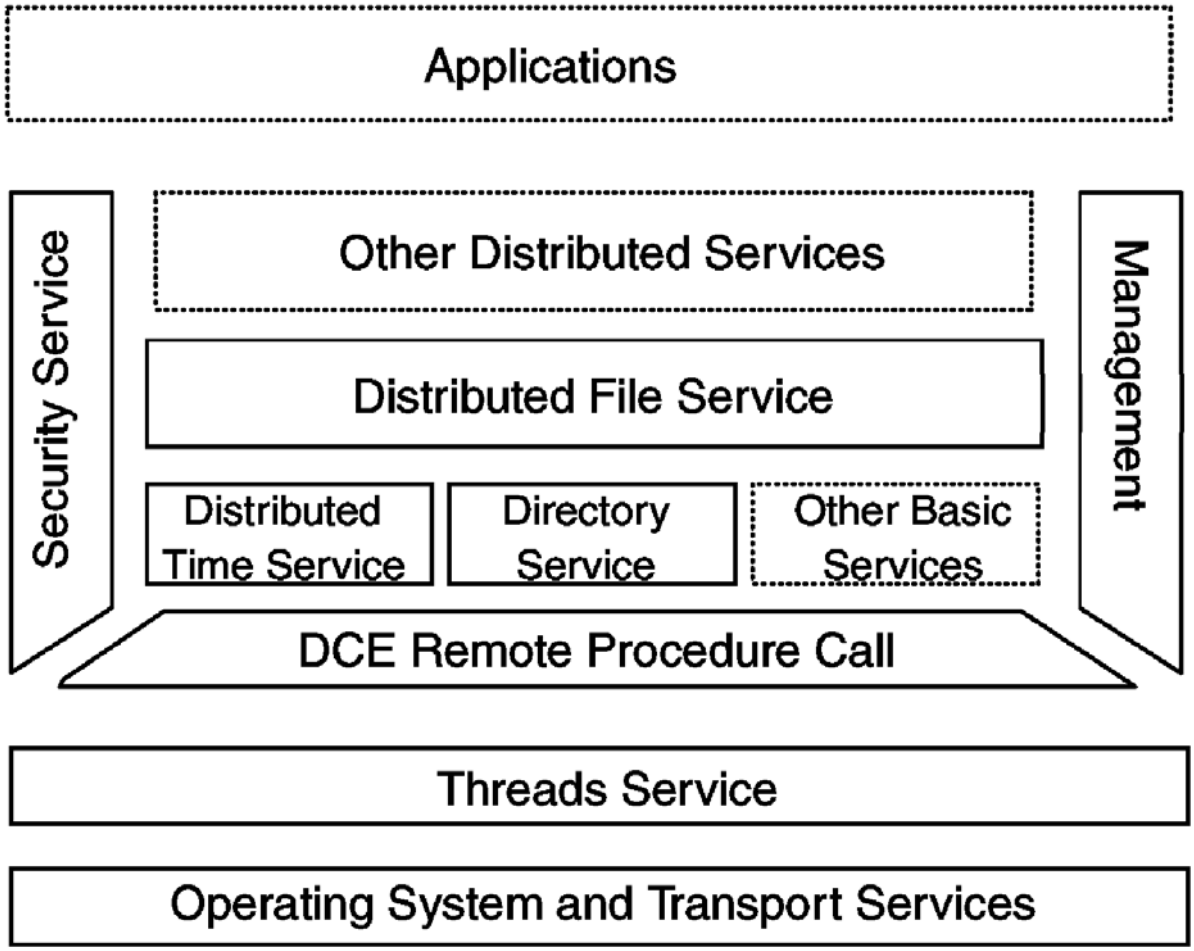
- **273 Deutschland**
- **1141 Deutschland Euro**
- **037 USA**
- **1047 USA**

**Für Entwicklungsaufgaben in C/C++ empfiehlt sich OS/390 Code Page 1047. Ein bekanntes Problem ist die unterschiedliche Darstellung der rechteckigen Klammern in Code Page 037 (normalerweise von OS/390 verwendet) und Code Page 1047.**

**Ein 3270 Klient (Emulator) konvertiert automatisch ASCII in EBCDIC. Die OS/390 Code Page läßt sich in der Regel im 3270 Klienten einstellen.**

| Term or concept | UNIX | z/OS |
|---|---|---|
| Start the operating system | Boot the system. | IPL (initial program load) the system. |
| Virtual storage given to each user of the system | Users get whatever virtual storage they need to reference, within the limits of the hardware and operating system. | Users each get an address space, a range of addresses extending to 2 GB (or even 16 EB) of virtual storage, though some of this storage contains system code that is common for all users. |
| Data storage | Files | Data sets (sometimes called files) |
| Data format | Byte orientation; organization of the data is provided by the application. | Record orientation; often an 80-byte record, reflecting the traditional punched card image. |
| System configuration data | The /etc file system controls characteristics. | Parameters in PARMLIB control how the system IPLs and how address spaces behave. |
| Scripting languages | Shell scripts, Perl, awk, and other languages | CLISTS (command lists) and REXX execs |
| Smallest element that performs work | A thread. The kernel supports multiple threads. | A task or a service request block (SRB). The z/OS base control program (BCP) supports multiple tasks and SRBs. |
| A long-running unit of work | A daemon | A started task or a long-running job; often this is a subsystem of z/OS. |
| Order in which the system searches for programs to run | Programs are loaded from the file system according to the user's PATH environmental variable (a list of directories to be searched). | The system searches the following libraries for the program to be loaded: TASKLIB, STEPLIB, JOBLIB, LPALST, and the linklist. |

| Term or concept | UNIX | z/OS |
|---|---|---|
| Using the system interactively | Users *log in* to systems and execute shell sessions in the shell environment. They can issue the **rlogin** or **telnet** commands to connect to the system. Each user can have many login sessions open at once. | Users *log on* to the system through TSO/E and its panel-driven interface, ISPF. A user ID is limited to having only one TSO/E logon session active at a time.<br><br>Users can also login to a z/OS UNIX shell environment using telnet, rlogin, or ssh. |
| Editing data or code | Many editors exist, such as vi, ed, sed, and emacs. | ISPF editor[a] |
| Source and destination for input and output data | stdin and stdout | SYSIN and SYSOUT<br>► SYSUT1 and SYSUT2 are used for utilities.<br>► SYSTSIN and SYSTSPRT are used for TSO/E users. |
| Managing programs | The **ps** shell command allows users to view processes and threads, and kill jobs with the **kill** command. | SDSF allows users to view and terminate their jobs. |

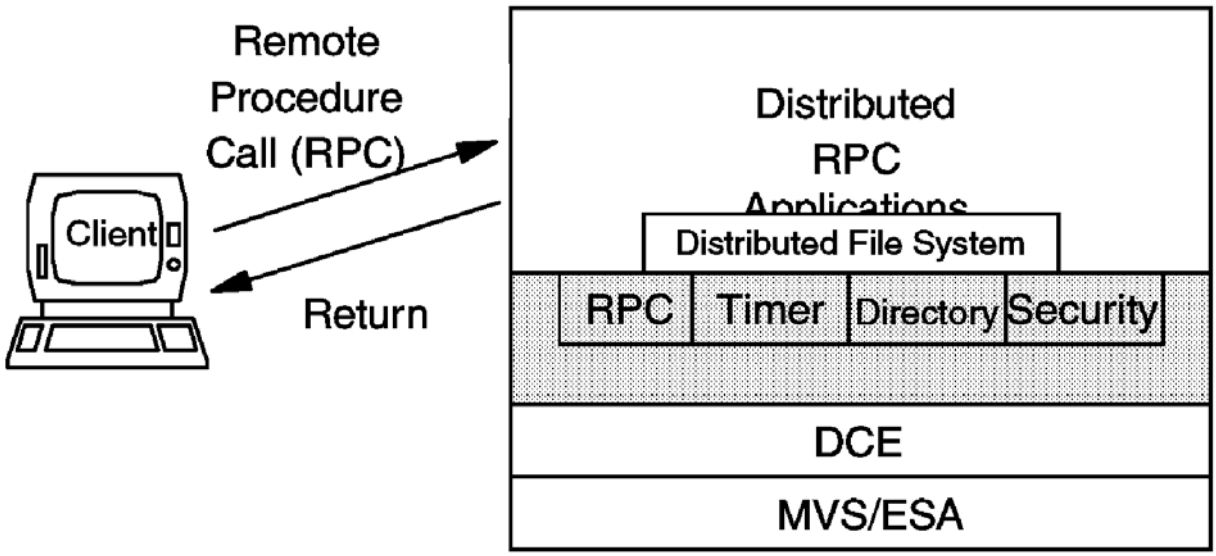



Distributed Computing Environment

**OSF's Distributed Computing Environment (DCE) is a comprehensive, integrated set of services and technologies that support the development, use and maintenance of distributed applications.**

**OSF/DCE offerings are organized in the following categories: Fundamental Distributed Services:**

I. **Remote Procedure Call**
II. **Directory Service**
III. **Time Service**
IV. **Security Service**
V. **Threads Service**
VI. **Data Sharing Services:**
VII. **Distributed File Service**




**The Distributed Computing Environment (DCE) is a package of services to simplify the development and execution of applications distributed over many heterogenous platforms.**

**In a distributed computing environment users do not see or know that they are accessing multiple systems, nor do they need to know addressing information about the resources they use. The users see only a single interface, whether the function is running locally or remote.**

**The services consist of the Remote Procedure Call support to link applications together, and the distributed services of directory, security, time, and a distributed file system.**

**The distributed file system allows hierarchical data resident on any of the platforms to be viewed by the user as one large file system. Services to distribute the data, replicate the data, and back up the data are also included.**