Processes and Threads

Chapter 2

The Process Model

A process is just an instance of an executing program, including the current values of the program counter (PC), registers, and variables.

The Process Model

Example:

Code

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    printf(" Preocess ID : %ld \n", getpid());
    sleep(100);
    return 0;
}
```

Location

Name	Date modified	Type	Size
main.c	2020-07-19 3:09 PM	C File	1 KB
main.exe	2020-07-19 3:06 PM	Application	164 KB

Processes

```
OS:>>
?(1)───mintty(1351)───bash(1352)─┬─main(1477)
                                 ├─main(1475)
                                 └─main(1476)
```

Process Name

Process ID

The Process Model

```c
#include <stdio.h>
int main()
{
    int sum = 0;
    for(int i=0; i<5; i++){
        sum += i;
    }
    return 0;
}
```

The Process Model

```c
#include <stdio.h>
int main()
{
    int sum = 0;
    for(int i=0; i<5; i++){
        sum += i;
    }
    return 0;
}
```

Assembly code

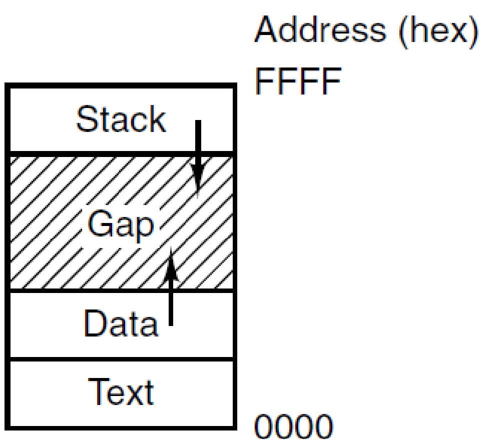
Machine code

Memory location

```
main:
          55
401102    push   rbp
          48 89 e5
401103    mov    rbp,rsp
          c7 45 fc 00 00 00 00
401106    mov    DWORD PTR [rbp-0x4],0x0
          c7 45 f8 00 00 00 00
40110d    mov    DWORD PTR [rbp-0x8],0x0
          83 7d f8 04
401114    cmp    DWORD PTR [rbp-0x8],0x4
          7f 0c
401118    jg     401126 <main+0x24>
          8b 45 f8
40111a    mov    eax,DWORD PTR [rbp-0x8]
          01 45 fc
40111d    add    DWORD PTR [rbp-0x4],eax
          83 45 f8 01
401120    add    DWORD PTR [rbp-0x8],0x1
          eb ee
401124    jmp    401114 <main+0x12>
          b8 00 00 00 00
401126    mov    eax,0x0
          5d
40112b    pop    rbp
          c3
40112c    ret
          0f 1f 00
40112d    nop    DWORD PTR [rax]
```

The Process Model

Processes in UNIX have their memory divided up into three segments: the text segment (i.e., the program code), the data segment (i.e., the variables), and the stack segment. The data segment grows upward and the stack grows downward.



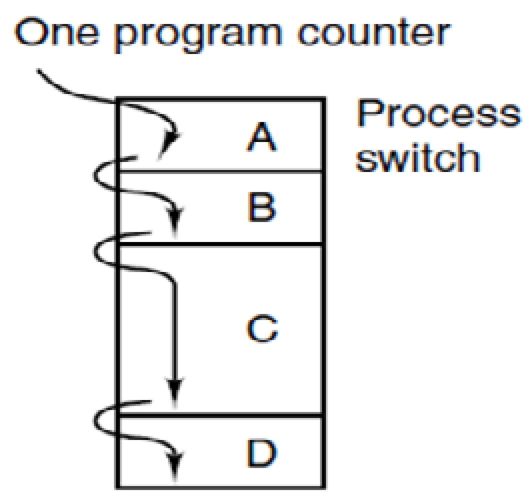
Processes have three segments: text, data, and stack.

The Process Model

Multiprogramming system

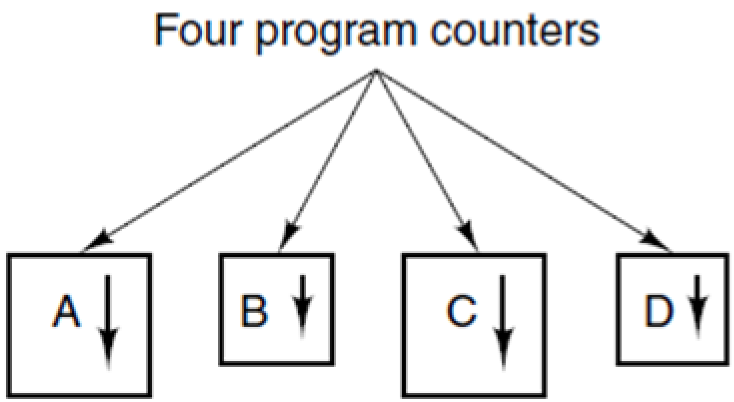
1. The CPU switches from process to process quickly.
2. The CPU runs each process for tens or hundreds of milliseconds.
3. Strictly speaking, at any one instant the CPU is running only one process.

The rapid switching back and forth is called **multiprogramming**.



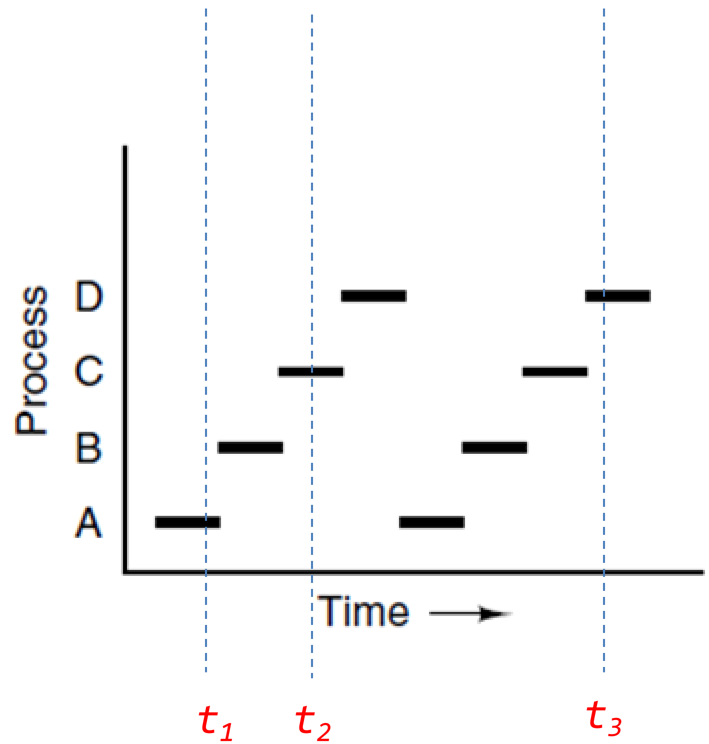
Multiprogramming of four programs

The Process Model



Conceptual model of
four independent, sequential processes.

The Process Model



Only one program is active at once.

Process Creation

Four principal events that cause processes to be created:

1. System initialization.
2. Execution of a process-creation system call by a running process.
3. A user request to create a new process.
4. Initiation of a batch job.

Process Creation

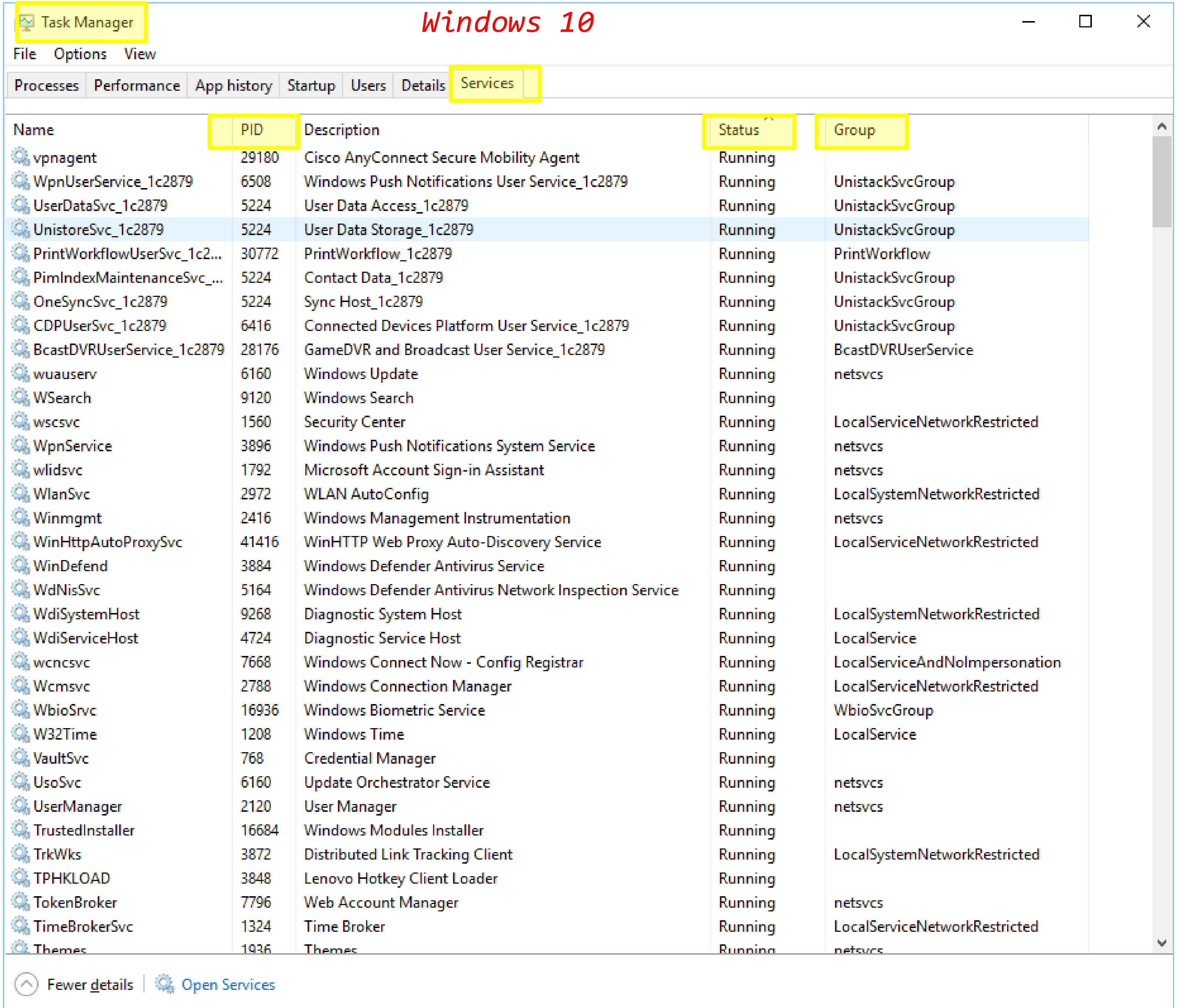
System initialization

When an operating system is booted, typically numerous processes are created.

1) Some of these are foreground processes, interacting with (human) users and perform work for them.

2) Others run in the background and are not associated with particular users, but instead have some specific function.

Processes that stay in the background to handle some activity such as email, Web pages, news, printing, and so on are called **daemons**.


Task Manager

Windows 10

File Options View

Processes | Performance | App history | Startup | Users | Details | Services

Name	PID	Description	Status	Group
vpnagent	29180	Cisco AnyConnect Secure Mobility Agent	Running	
WpnUserService_1c2879	6508	Windows Push Notifications User Service_1c2879	Running	UnistackSvcGroup
UserDataSvc_1c2879	5224	User Data Access_1c2879	Running	UnistackSvcGroup
UnistoreSvc_1c2879	5224	User Data Storage_1c2879	Running	UnistackSvcGroup
PrintWorkflowUserSvc_1c2...	30772	PrintWorkflow_1c2879	Running	PrintWorkflow
PimIndexMaintenanceSvc_...	5224	Contact Data_1c2879	Running	UnistackSvcGroup
OneSyncSvc_1c2879	5224	Sync Host_1c2879	Running	UnistackSvcGroup
CDPUserSvc_1c2879	6416	Connected Devices Platform User Service_1c2879	Running	UnistackSvcGroup
BcastDVRUserService_1c2879	28176	GameDVR and Broadcast User Service_1c2879	Running	BcastDVRUserService
wuauserv	6160	Windows Update	Running	netsvcs
WSearch	9120	Windows Search	Running	
wscsvc	1560	Security Center	Running	LocalServiceNetworkRestricted
WpnService	3896	Windows Push Notifications System Service	Running	netsvcs
wlidsvc	1792	Microsoft Account Sign-in Assistant	Running	netsvcs
WlanSvc	2972	WLAN AutoConfig	Running	LocalSystemNetworkRestricted
Winmgmt	2416	Windows Management Instrumentation	Running	netsvcs
WinHttpAutoProxySvc	41416	WinHTTP Web Proxy Auto-Discovery Service	Running	LocalServiceNetworkRestricted
WinDefend	3884	Windows Defender Antivirus Service	Running	
WdNisSvc	5164	Windows Defender Antivirus Network Inspection Service	Running	
WdiSystemHost	9268	Diagnostic System Host	Running	LocalSystemNetworkRestricted
WdiServiceHost	4724	Diagnostic Service Host	Running	LocalService
wcncsvc	7668	Windows Connect Now - Config Registrar	Running	LocalServiceAndNoImpersonation
Wcmsvc	2788	Windows Connection Manager	Running	LocalServiceNetworkRestricted
WbioSrvc	16936	Windows Biometric Service	Running	WbioSvcGroup
W32Time	1208	Windows Time	Running	LocalService
VaultSvc	768	Credential Manager	Running	
UsoSvc	6160	Update Orchestrator Service	Running	netsvcs
UserManager	2120	User Manager	Running	netsvcs
TrustedInstaller	16684	Windows Modules Installer	Running	
TrkWks	3872	Distributed Link Tracking Client	Running	LocalSystemNetworkRestricted
TPHKLOAD	3848	Lenovo Hotkey Client Loader	Running	
TokenBroker	7796	Web Account Manager	Running	netsvcs
TimeBrokerSvc	1324	Time Broker	Running	LocalServiceNetworkRestricted
Themes	1936	Themes	Running	netsvcs

Fewer details | Open Services

Ubuntu 18.04.4 LTS

ps -aux | less

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.1  0.1 225984  9744 ?        Ss   14:31   0:11 /sbin/init splash
root         2  0.0  0.0      0     0 ?        S    14:31   0:00 [kthreadd]
root         6  0.0  0.0      0     0 ?        I<   14:31   0:00 [mm_percpu_wq]
root         7  0.0  0.0      0     0 ?        S    14:31   0:00 [ksoftirqd/0]
root         8  0.4  0.0      0     0 ?        I    14:31   0:33 [rcu_sched]
root         9  0.0  0.0      0     0 ?        I    14:31   0:00 [rcu_bh]
root        10  0.0  0.0      0     0 ?        S    14:31   0:00 [migration/0]
root        11  0.0  0.0      0     0 ?        S    14:31   0:00 [watchdog/0]
root        12  0.0  0.0      0     0 ?        S    14:31   0:00 [cpuhp/0]
root        13  0.0  0.0      0     0 ?        S    14:31   0:00 [cpuhp/1]
root        14  0.0  0.0      0     0 ?        S    14:31   0:00 [watchdog/1]
root        15  0.0  0.0      0     0 ?        S    14:31   0:00 [migration/1]
root        16  0.0  0.0      0     0 ?        S    14:31   0:03 [ksoftirqd/1]
root        19  0.0  0.0      0     0 ?        S    14:31   0:00 [cpuhp/2]
root        20  0.0  0.0      0     0 ?        S    14:31   0:00 [watchdog/2]
root        21  0.0  0.0      0     0 ?        S    14:31   0:00 [migration/2]
root        22  0.0  0.0      0     0 ?        S    14:31   0:00 [ksoftirqd/2]
root        24  0.0  0.0      0     0 ?        I<   14:31   0:00 [kworker/2:0H]
root        25  0.0  0.0      0     0 ?        S    14:31   0:00 [cpuhp/3]
root        26  0.0  0.0      0     0 ?        S    14:31   0:00 [watchdog/3]
root        27  0.0  0.0      0     0 ?        S    14:31   0:00 [migration/3]
root        28  0.0  0.0      0     0 ?        S    14:31   0:00 [ksoftirqd/3]
root        30  0.0  0.0      0     0 ?        I<   14:31   0:00 [kworker/3:0H]
root        31  0.0  0.0      0     0 ?        S    14:31   0:00 [kdevtmpfs]
root        32  0.0  0.0      0     0 ?        I<   14:31   0:00 [netns]
root        33  0.0  0.0      0     0 ?        S    14:31   0:00 [rcu_tasks_kthre]
root        34  0.0  0.0      0     0 ?        S    14:31   0:00 [kauditd]
root        39  0.0  0.0      0     0 ?        S    14:31   0:00 [khungtaskd]
root        40  0.0  0.0      0     0 ?        S    14:31   0:00 [oom_reaper]
root        41  0.0  0.0      0     0 ?        I<   14:31   0:00 [writeback]
root        42  0.0  0.0      0     0 ?        S    14:31   0:00 [kcompactd0]
root        43  0.0  0.0      0     0 ?        SN   14:31   0:00 [ksmd]
root        44  0.0  0.0      0     0 ?        SN   14:31   0:00 [khugepaged]
root        45  0.0  0.0      0     0 ?        I<   14:31   0:00 [crypto]
```

Process Creation

Execution of a process creation system call
by a running process

Often a running process will issue system calls to create one or more new processes to help it do its job.

For example, if a large amount of data is being fetched over a network for subsequent processing, it may be convenient to create one process to fetch the data and put them in a shared buffer while a second process removes the data items and processes them. On a multiprocessor, allowing each process to run on a different CPU may also make the job go faster.

Process Creation

A user request to create a new process

In interactive systems, users can start a program by typing a command or (double) clicking on an icon. Taking either of these actions starts a new process and runs the selected program in it.

In command-based UNIX systems running X, the new process takes over the window in which it was started.

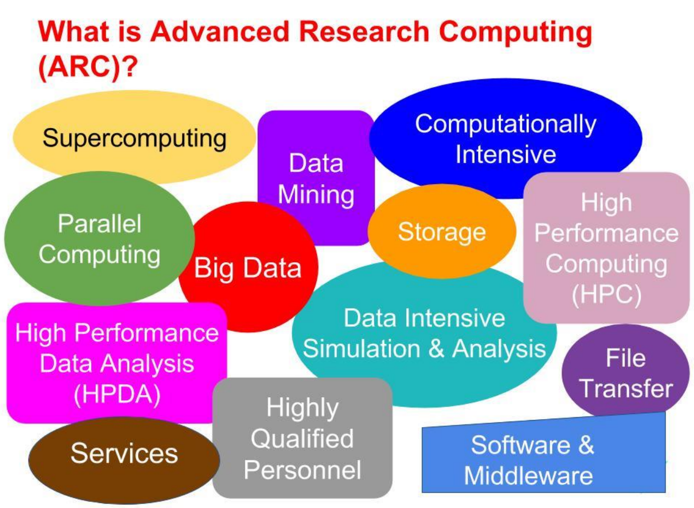
```
OS:>> ls
main.c  main.exe
OS:>> ./main.exe
 Preocess ID : 1487
OS:>>
```

Process Creation

Initiation of a batch job

The last situation in which processes are created applies only to the batch systems found on large mainframes. Think of inventory management at the end of a day at a chain of stores. Here users can submit batch jobs to the system (possibly remotely).

When the operating system decides that it has the resources to run another job, it creates a new process and runs the next job from the input queue in it.



https://alliancecan.ca/en

Process Termination

Typical conditions which terminate a process:

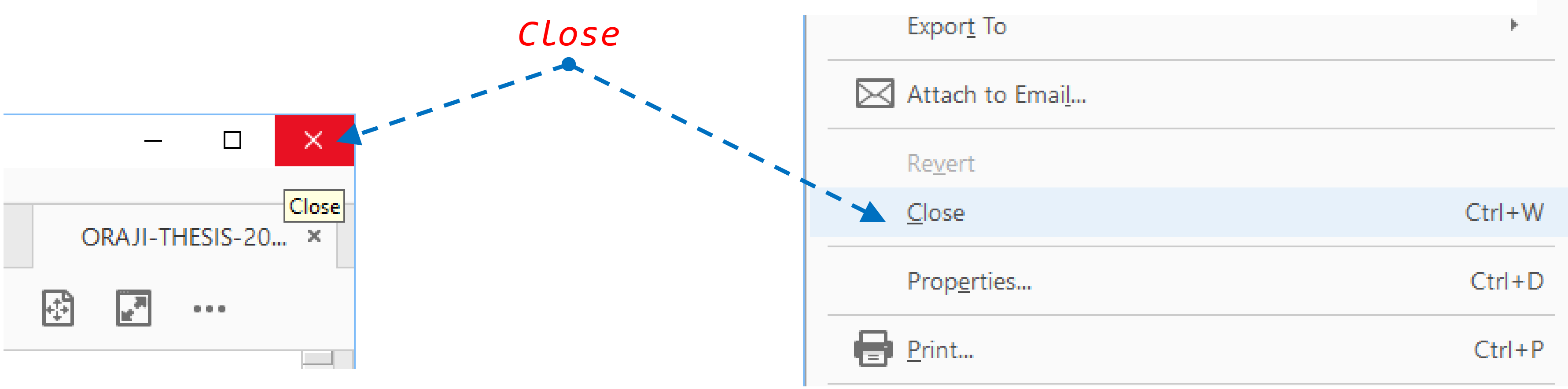
1. Normal exit (voluntary).
2. Error exit (voluntary).
3. Fatal error (involuntary).
4. Killed by another process (involuntary).

Process Termination

Normal exit (voluntary)

Most processes terminate because they have done their work. When a compiler has compiled the program given to it, the compiler executes a system call to tell the operating system that it is finished. This call is exit in UNIX and ExitProcess in Windows.

Screen-oriented programs also support voluntary termination. Word processors, Internet browsers, and similar programs always have an icon or menu item that the user can click to tell the process to remove any temporary files it has open and then terminate.



Process Termination

Error exit (voluntary)

The second reason for termination is that the process discovers a fatal error. For example, if a user types the command
cc foo.c
to compile the program *foo.c* and no such file exists, the compiler simply announces this fact and exits.

```
~$ gcc foo.c
gcc: error: foo.c: No such file or directory
gcc: fatal error: no input files
compilation terminated.
~$
```

Screen-oriented interactive processes generally do not exit when given bad parameters. Instead they pop up a dialog box and ask the user to try again.

Process Termination

Fatal error (involuntary)

The third reason for termination is an error caused by the process, often due to a program bug. Examples include executing an illegal instruction, referencing nonexistent memory, or dividing by zero.

In some systems (e.g., UNIX), a process can tell the operating system that it wishes to handle certain errors itself, in which case the process is signaled (interrupted) instead of terminated when one of the errors occurs.

Process Termination

Killed by another process (involuntary)

The fourth reason a process might terminate is that the process executes a system call telling the operating system to kill some other process. In UNIX this call is kill. The corresponding Win32 function is TerminateProcess. In both cases, the killer must have the necessary authorization to do in the killee.

```
OS:>> ./program.exe  &
[1] 1243
OS:>>  Process waitng 30 seconds.

OS:>> pstree -p
?(1)───mintty(1169)───bash(1170)─┬─program(1243)
                                 └─pstree(1244)
OS:>> kill 1243
OS:>> fg
-bash: fg: job has terminated
[1]+  Terminated              ./program.exe
OS:>>
```

Process Hierarchies

In some systems, when a process creates another process, the parent process and child process continue to be associated in certain ways. The child process can itself create more processes.

Process Hierarchies
UNIX

In UNIX, a process and all of its children and further descendants together form a process group.

When a user sends a signal from the keyboard, the signal is delivered to all members of the process group currently associated with the keyboard.

Individually, each process can catch the signal, ignore the signal, or take the default action, which is to be killed by the signal.

Process Hierarchies Windows

In contrast, Windows has no concept of a process hierarchy. All processes are equal.

The only hint of a process hierarchy is that when a process is created, the parent is given a special token (called a **handle**) that it can use to control the child.

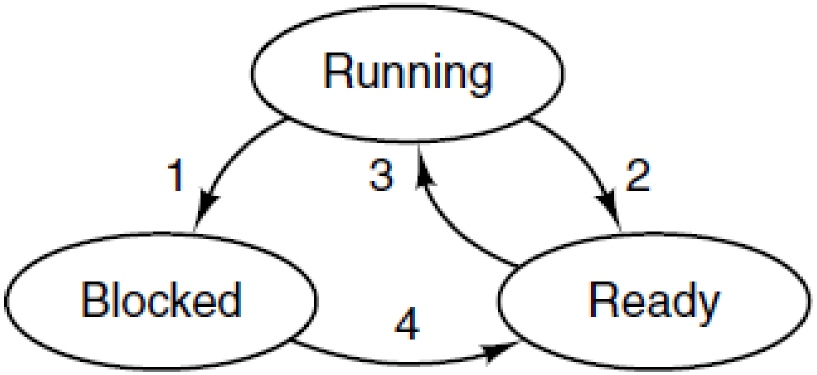
However, it is free to pass this token to some other process, thus invalidating the hierarchy. Processes in UNIX cannot disinherit their children.

Process States

Three states a process may be in:

1. Running (actually using the CPU at that instant)
2. Ready (scheduled and runnable; temporarily stopped to let another process run)
3. Blocked (unable to run until some external event happens)

Process States



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

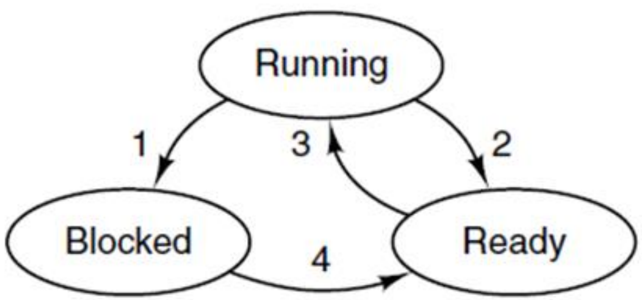
A process can be in running, blocked, or ready state. Transitions between these states are as shown.

Process States

Transition 1 occurs when the operating system discovers that a process cannot continue right now.

In some systems the process can execute a system call, such as pause, to get into blocked state.

In other systems, including UNIX, when a process reads from a pipe or special file (e.g., a terminal) and there is no input available, the process is automatically blocked.




```
OS:>>
OS:>> cat data.txt | tr '[A-Z]' 'a-z'| tr -C '[a-z]' '\n' | grep cow
cow
cows
cows
```

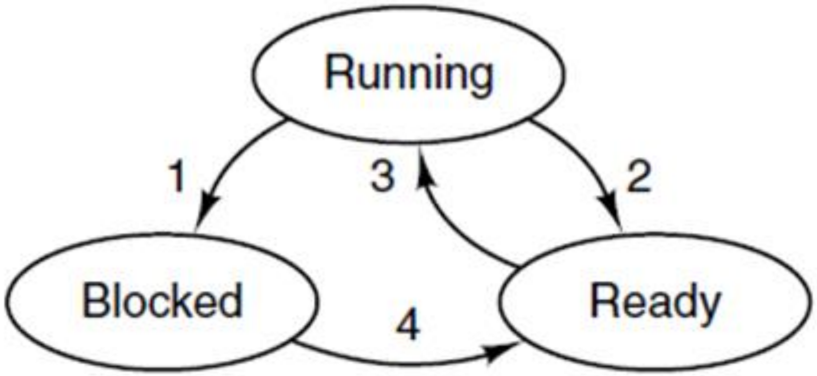
1 2 3 4

Process States

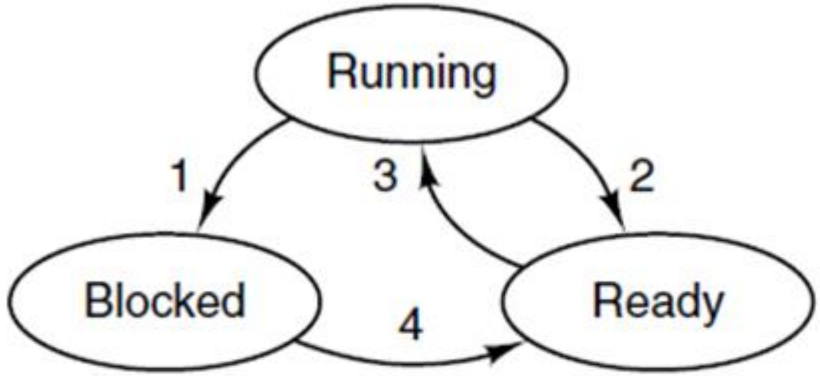
Transitions 2 and 3 are caused by the process scheduler, a part of the operating system, without the process even knowing about them.

Transition 2 occurs when the scheduler decides that the running process has run long enough, and it is time to let another process have some CPU time.

Transition 3 occurs when all the other processes have had their fair share and it is time for the first process to get the CPU to run again.



Process States



Transition 4 occurs when the external event for which a process was waiting (such as the arrival of some input) happens.

If no other process is running at that instant, transition 3 will be triggered and the process will start running.

Otherwise it may have to wait in *ready* state for a little while until the CPU is available and its turn comes.

Implementation of Processes

To implement the process model, the operating system maintains a table (an array of structures), called the **process table**, with one entry per process.

This entry contains important information about the process' state, including its program counter, stack pointer, memory allocation, the status of its open files, its accounting and scheduling information, and everything else about the process that must be saved when the process is switched from running to ready or blocked state so that it can be restarted later as if it had never been stopped.

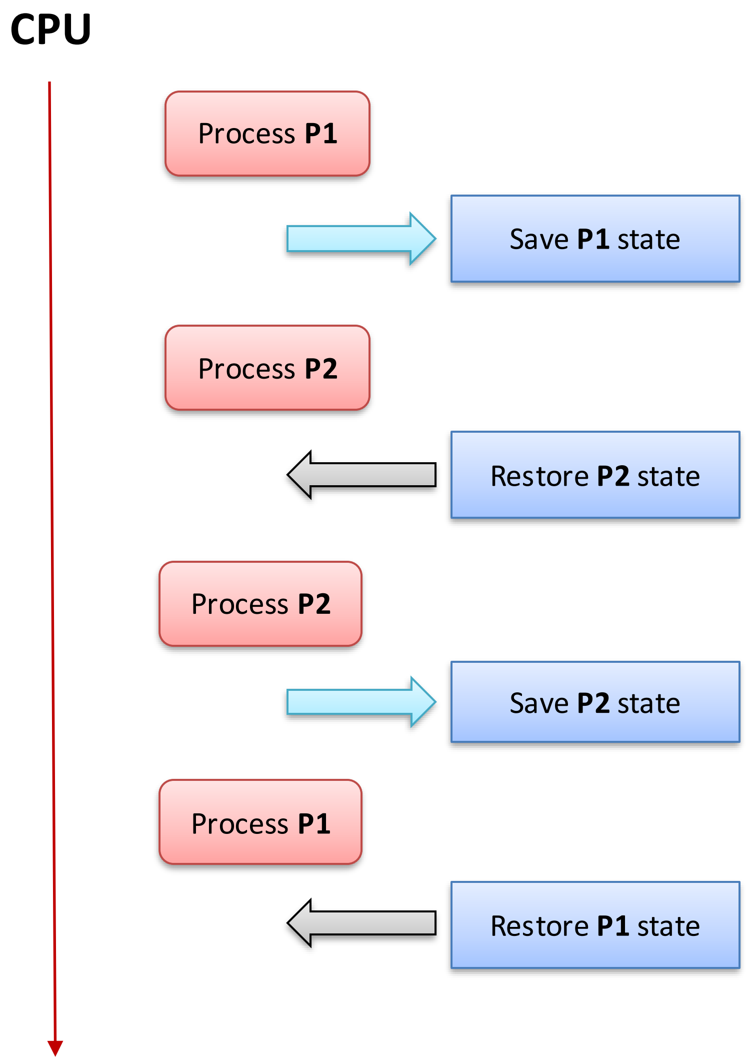
Implementation of Processes
Context Switch

A context switch refers to the process of saving the current state of a running process and loading the saved state of another process so that it can resume execution.

When the CPU scheduler decides to switch from one process to another, it must save the current process's context, including its program counter, register values, and other relevant state information, into the process's PCB (Process Control Block).

PCB is a data structure that represents a single process in the operating system. The PCB is a crucial component within the process table.

Implementation of Processes
Context Switch



Implementation of Processes
Context Switch

```c
#include <stdio.h>
#include <ucontext.h>
#include <unistd.h>
#define REG_RIP 16

/* user-level context (uc), includes
   1) signal mask,
   2) stack info,
   3) link to next context,
   and an embedded machine-level context (mc).

   Machine-level context (mc) contains
   General-purpose registers: RIP, RSP, RAX, etc.
   . . . . . . .
*/
int main(int argc, const char *argv[]){
    ucontext_t ctx;
    printf("Capturing context...\n");
    getcontext(&ctx);
    printf("Context captured. Ready to inspect.\n");
    printf("RIP: %llx\n", (unsigned long long)ctx.uc_mcontext.gregs[REG_RIP]);
    return 0;
}
```

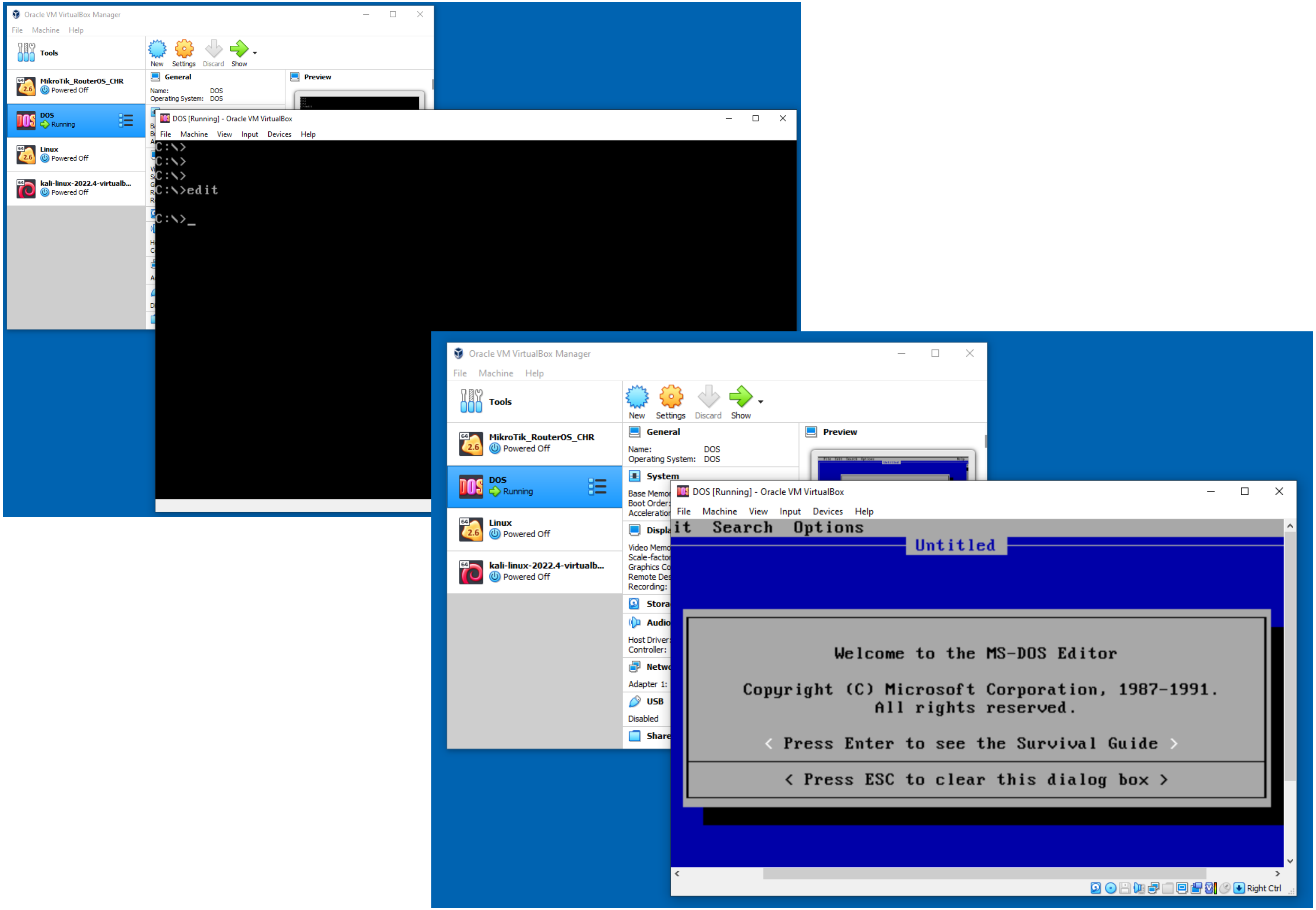
```
Capturing context...
Context captured. Ready to inspect.
RIP: 56f2d961c1f2
```

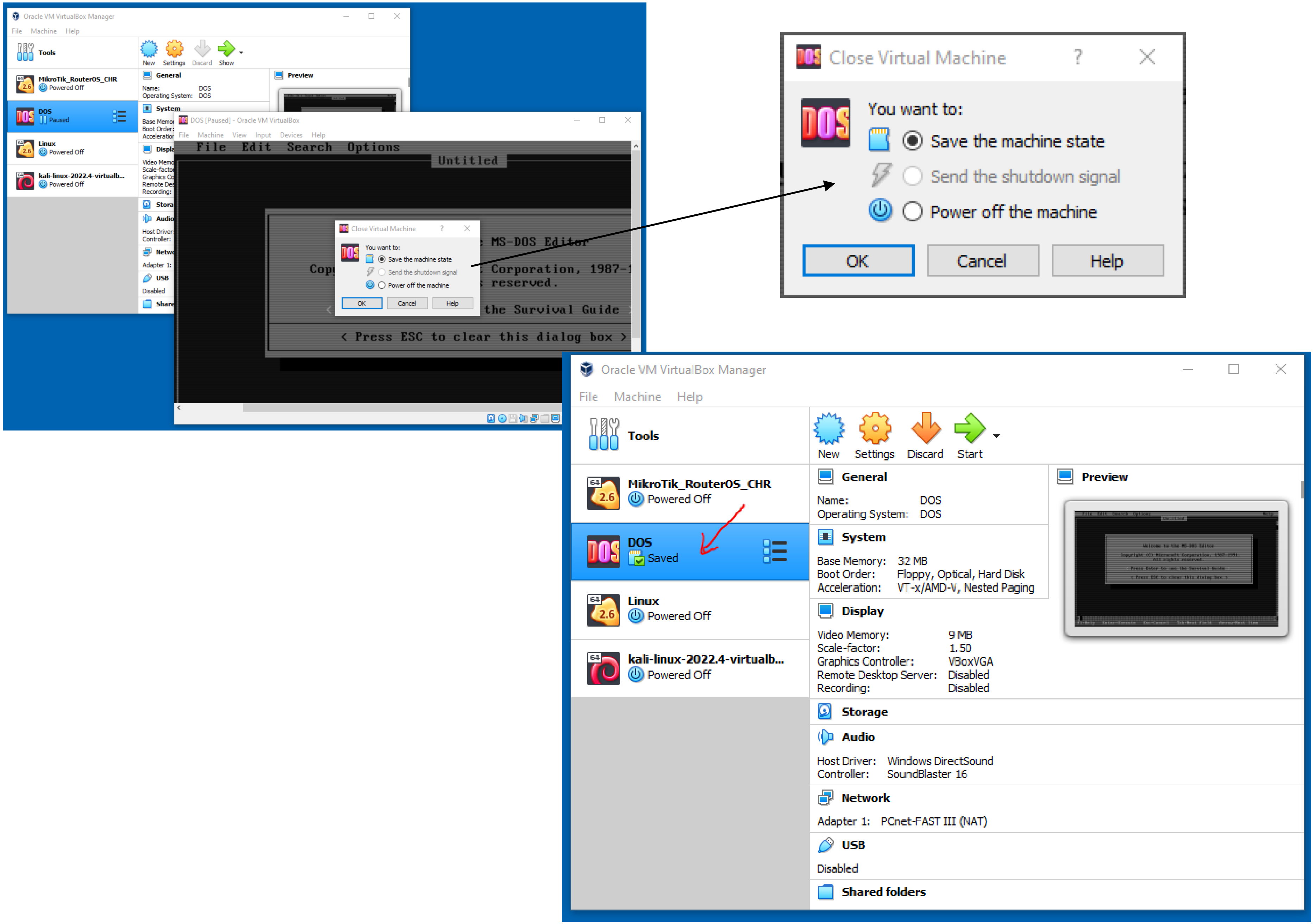
Implementation of Processes
Context Switch

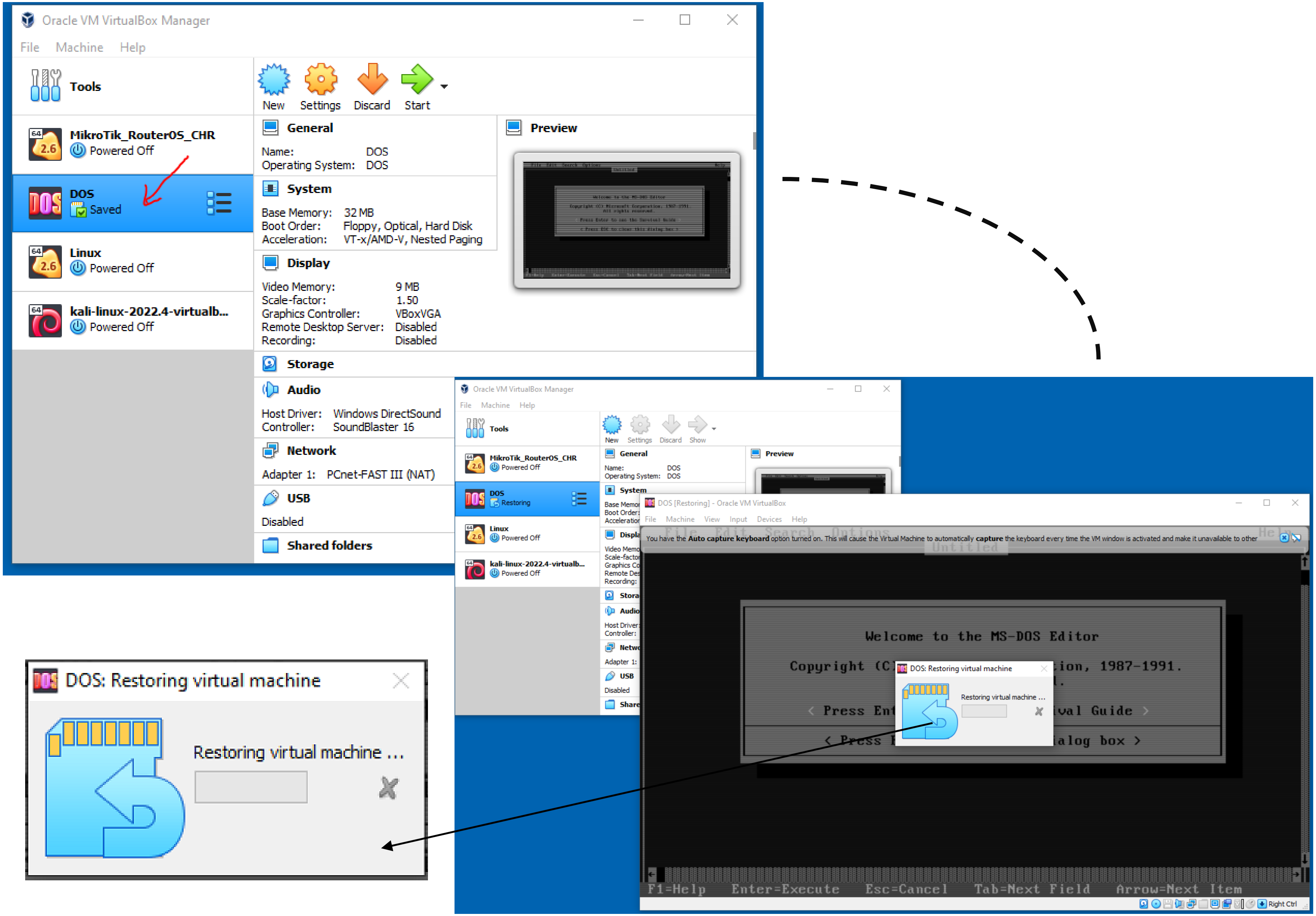
Implementation of Processes
Context Switch

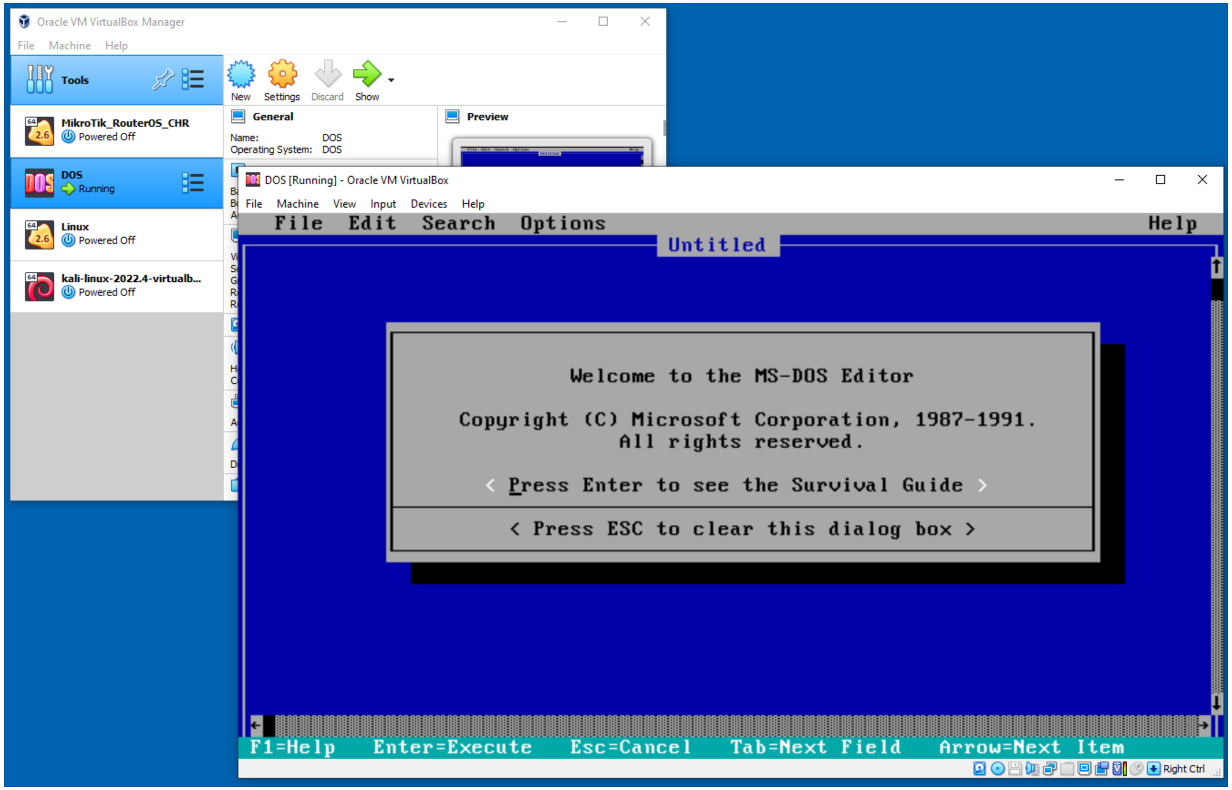
```c
#include <stdio.h>
#include <ucontext.h>
#include <unistd.h>

int main(){
	ucontext_t context;

	getcontext(&context);
	puts("Hello OS ");
	sleep(1);
	setcontext(&context);
	return 0;
}
```

```
Hello OS
Hello OS
Hello OS
Hello OS
Hello OS
Hello OS
```


Oracle VM VirtualBox Manager
File Machine Help
Tools
New Settings Discard Show
MikroTik_RouterOS_CHR
Powered Off
DOS
Running
Linux
Powered Off
kali-linux-2022.4-virtualb...
Powered Off
General
Name: DOS
Operating System: DOS
Preview
DOS [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
C:\>
C:\>
C:\>
C:\>edit
C:\>_
System
Storage
Audio
USB
Disabled
it Search Options
Untitled
Welcome to the MS-DOS Editor
Copyright (C) Microsoft Corporation, 1987-1991.
All rights reserved.
< Press Enter to see the Survival Guide >
< Press ESC to clear this dialog box >
Right Ctrl


Close Virtual Machine
You want to:
Save the machine state
Send the shutdown signal
Power off the machine
OK
Cancel
Help
Oracle VM VirtualBox Manager
File
Machine
Help
Tools
New
Settings
Discard
Start
MikroTik_RouterOS_CHR
Powered Off
DOS
Saved
Linux
Powered Off
kali-linux-2022.4-virtualb...
Powered Off
General
Name: DOS
Operating System: DOS
System
Base Memory: 32 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging
Display
Video Memory: 9 MB
Scale-factor: 1.50
Graphics Controller: VBoxVGA
Remote Desktop Server: Disabled
Recording: Disabled
Storage
Audio
Host Driver: Windows DirectSound
Controller: SoundBlaster 16
Network
Adapter 1: PCnet-FAST III (NAT)
USB
Disabled
Shared folders
Preview
DOS [Paused] - Oracle VM VirtualBox
File Edit Search Options
Untitled
MS-DOS Editor
Corporation, 1987-1
reserved.
the Survival Guide
Press ESC to clear this dialog box


Oracle VM VirtualBox Manager
File Machine Help
Tools
New Settings Discard Start
MikroTik_RouterOS_CHR
Powered Off
DOS
Saved
Linux
Powered Off
kali-linux-2022.4-virtualb...
Powered Off
General
Name: DOS
Operating System: DOS
System
Base Memory: 32 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging
Display
Video Memory: 9 MB
Scale-factor: 1.50
Graphics Controller: VBoxVGA
Remote Desktop Server: Disabled
Recording: Disabled
Storage
Audio
Host Driver: Windows DirectSound
Controller: SoundBlaster 16
Network
Adapter 1: PCnet-FAST III (NAT)
USB
Disabled
Shared folders
Preview
DOS: Restoring virtual machine
Restoring virtual machine ...
DOS [Restoring] - Oracle VM VirtualBox
File Machine View Input Devices Help
Restoring
Show
Welcome to the MS-DOS Editor
F1=Help Enter=Execute Esc=Cancel Tab=Next Field Arrow=Next Item
Right Ctrl


Oracle VM VirtualBox Manager
File Machine Help
Tools
New Settings Discard Show
MikroTik_RouterOS_CHR
Powered Off
DOS
Running
Linux
Powered Off
kali-linux-2022.4-virtualb...
Powered Off
General
Name: DOS
Operating System: DOS
Preview
DOS [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
File Edit Search Options Help
Untitled
Welcome to the MS-DOS Editor
Copyright (C) Microsoft Corporation, 1987-1991.
All rights reserved.
< Press Enter to see the Survival Guide >
< Press ESC to clear this dialog box >
F1=Help Enter=Execute Esc=Cancel Tab=Next Field Arrow=Next Item
Right Ctrl

Implementation of Processes

Process management	Memory management	File management
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Some of the fields of a typical process table entry.

Implementation of Processes

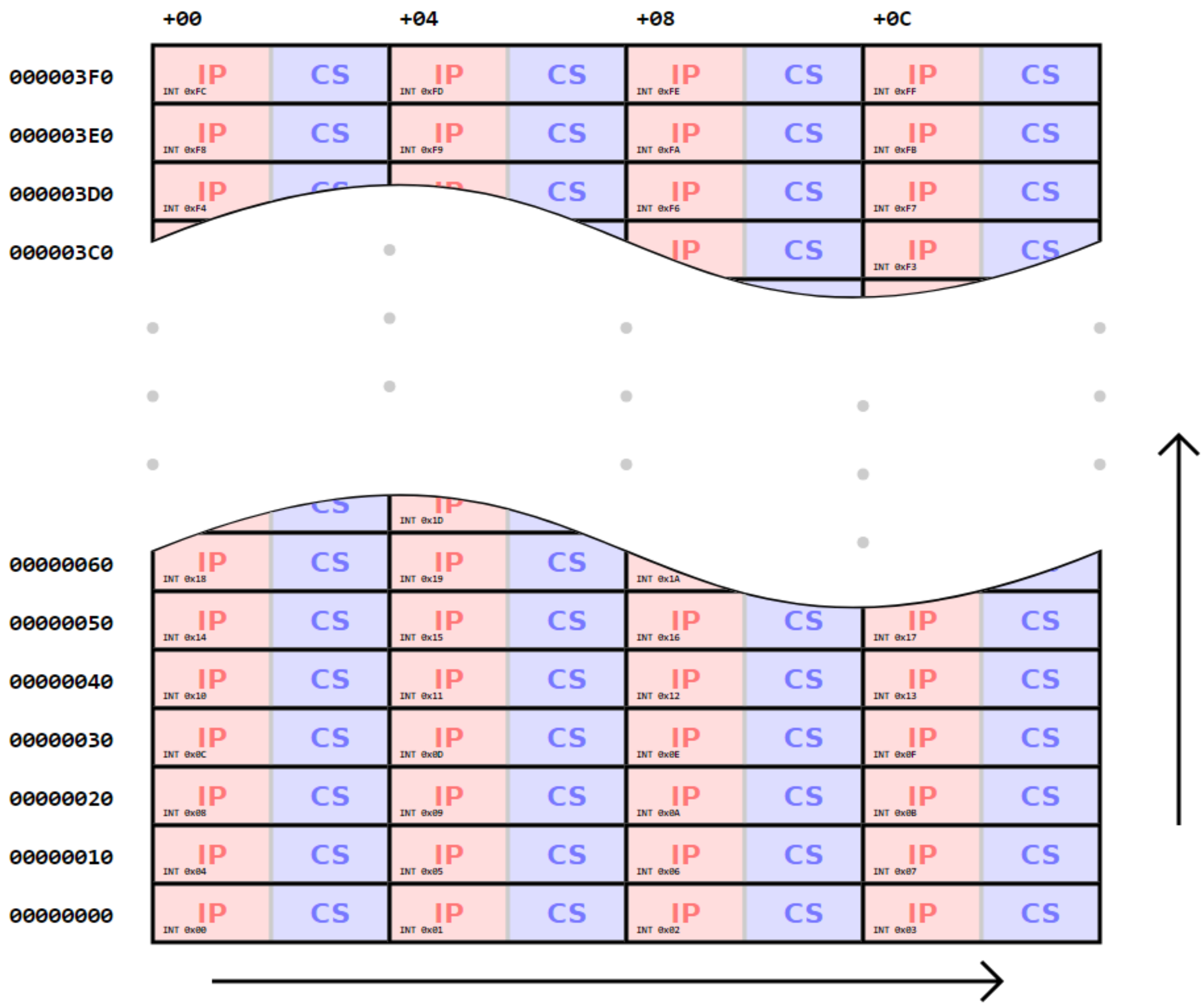
The illusion of multiple sequential processes maintained on one (or each) CPU. Suppose that user process is running when a disk interrupt happens.

Skeleton of what the lowest level of the operating system does when an interrupt occurs.

1. Hardware stacks program counter, etc.
2. Hardware loads new program counter from interrupt vector.
3. Assembly language procedure saves registers.
4. Assembly language procedure sets up new stack.
5. C interrupt service runs (typically reads and buffers input).
6. Scheduler decides which process is to run next.
7. C procedure returns to the assembly code.
8. Assembly language procedure starts up new current process.

Associated with each I/O class is a location (typically at a fixed location near the bottom of memory) called the **interrupt vector**.

Interrupt Vector Table



Interrupt Vector Table of x86 processors running in real mode.

Source: WIKIPEDIA

Modeling Multiprogramming

CPU utilization

a probabilistic viewpoint

Suppose that a process spends a fraction p of its time waiting for I/O to complete. With n processes in memory at once, the probability that all n processes are waiting for I/O (in which case the CPU will be idle) is p^n. The CPU utilization is then given by the formula:

$$\text{CPU utilization} = 1 - p^n$$

Modeling Multiprogramming

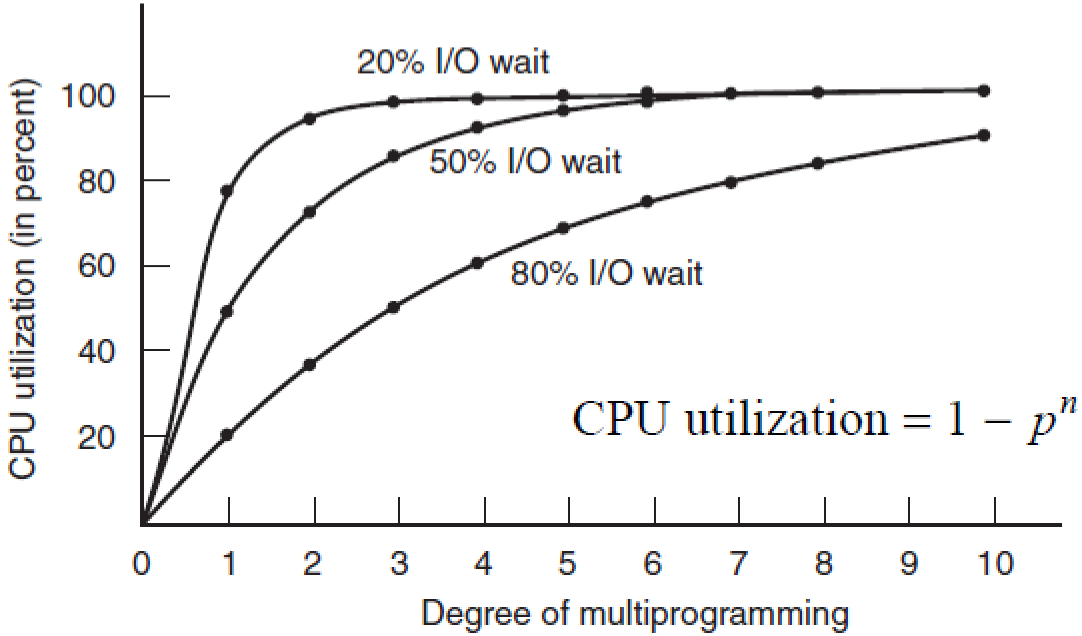
	P1	P2	P3		
0 idle	1-p	1-p	1-p	$(1-p)^3$	
1 idle	p	1-p	1-p	$p(1-p)^2$	
	1-p	p	1-p	$p(1-p)^2$	$3p(1-p)^2$
	1-p	1-p	p	$p(1-p)^2$	
2 idle	1-p	p	p	$p^2(1-p)$	
	p	1-p	p	$p^2(1-p)$	$3p^2(1-p)$
	p	p	1-p	$p^2(1-p)$	
3 idle	p	p	p	p^3	

CPU - utilization

$$U = (1-p)^3 + 3p(1-p)^2 + 3p^2(1-p)$$
$$U = 1 - p^3$$

Modeling Multiprogramming

CPU utilization
a probabilistic viewpoint



Modeling Multiprogramming

CPU utilization

a probabilistic viewpoint

From the figure it is clear that if processes spend 80% of their time waiting for I/O, at least 10 processes must be in memory at once to get the CPU waste below 10%.

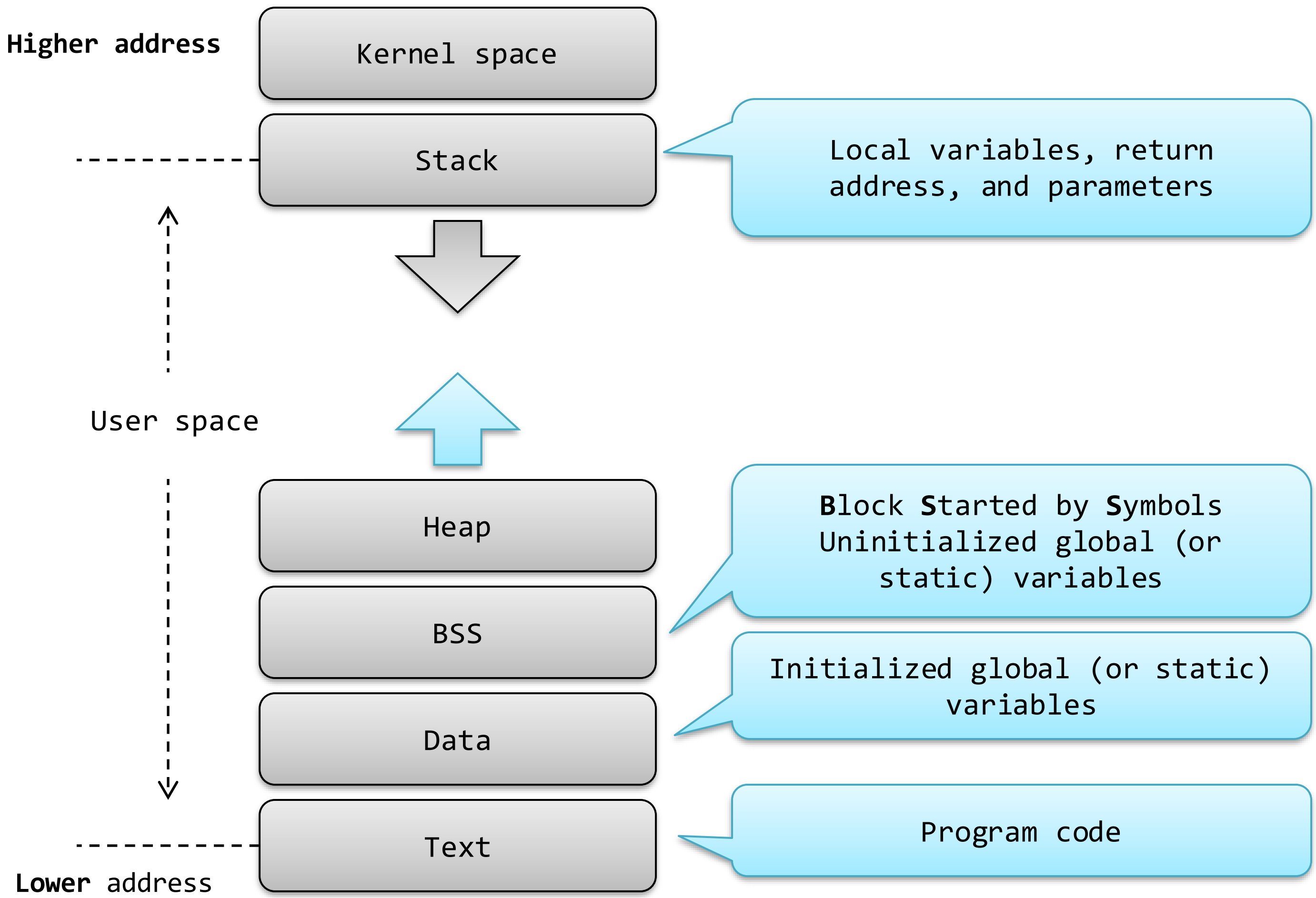
CPU waste = $(0.8)^{10} \approx 0.1074$

When you realize that an interactive process waiting for a user to type something at a terminal (or click on an icon) is in I/O wait state, it should be clear that I/O wait times of 80% and more are not unusual.

It should be pointed out that the probabilistic model assumes that all *n* processes are independent.

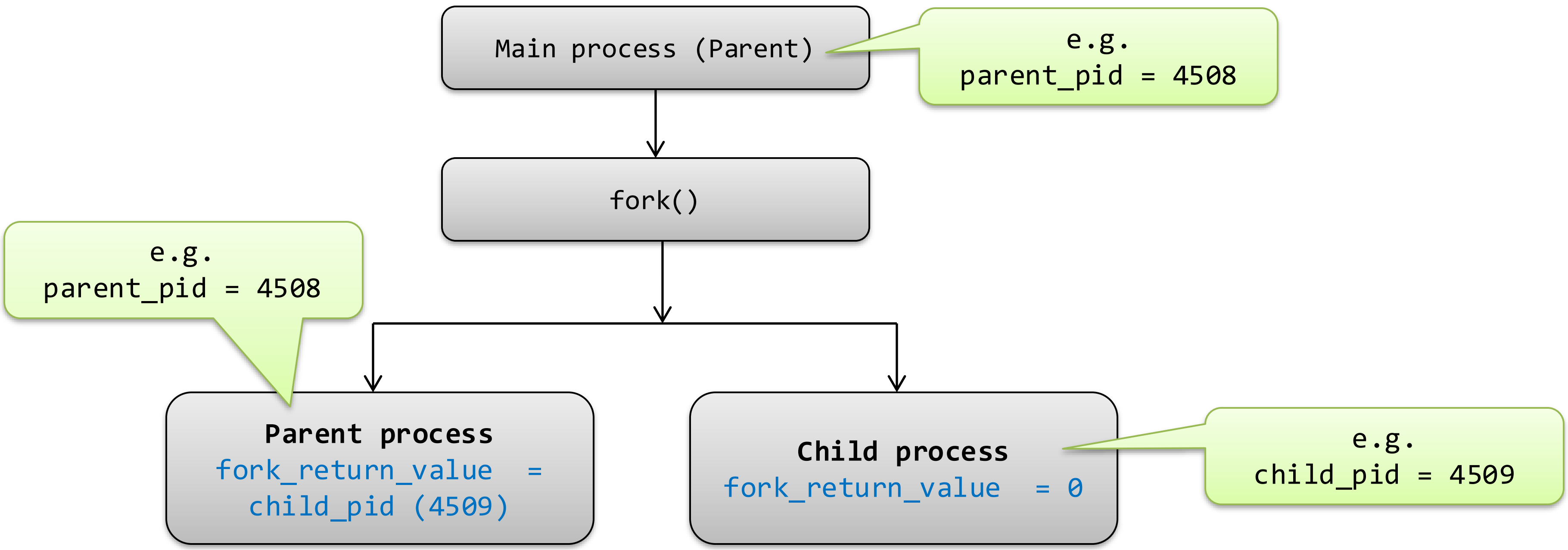
Linux

Processes in Unix



Linux

Processes creation



Linux

Processes creation

```
#define TRUE 1

while (TRUE) {                                   /* repeat forever */
      type_prompt( );                            /* display prompt on the screen */
      read_command(command, parameters);         /* read input from terminal */

      if (fork( ) != 0) {                        /* fork off child process */
          /* Parent code. */
          waitpid(−1, &status, 0);               /* wait for child to exit */
      } else {
          /* Child code. */
          execve(command, parameters, 0);        /* execute command */
      }
}
```

A stripped-down shell. Throughout this book, *TRUE* is assumed to be defined as 1.

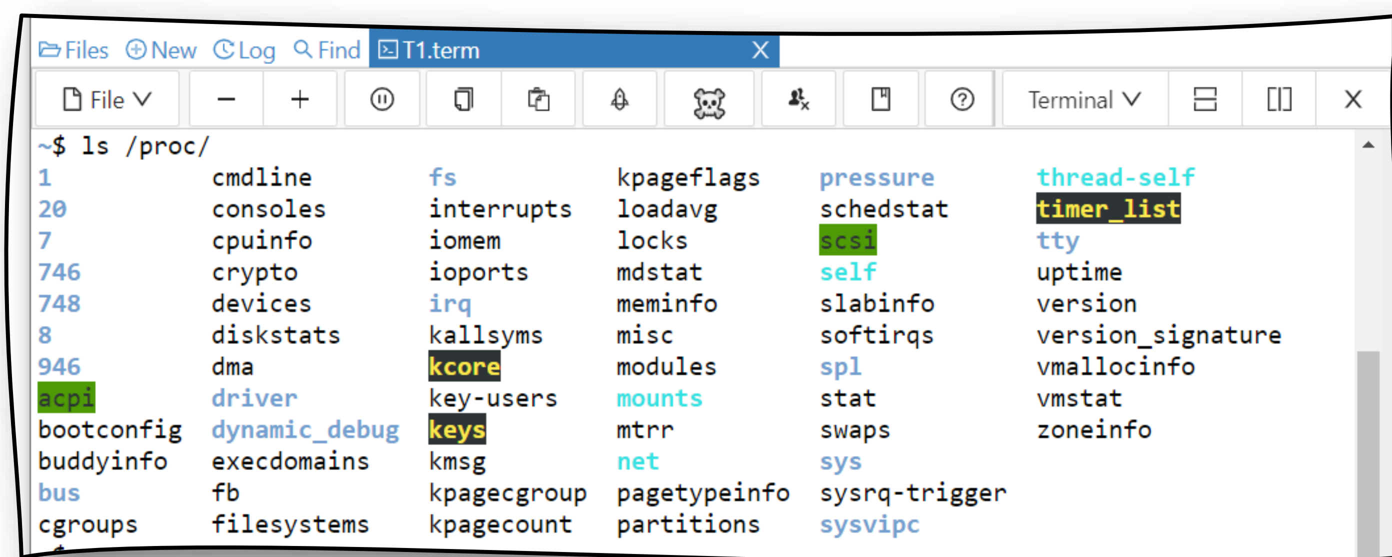
Linux

process information pseudo-filesystem (proc)

man proc

The proc filesystem is a pseudo-filesystem which provides an interface to kernel data structures. It is commonly mounted at /proc.

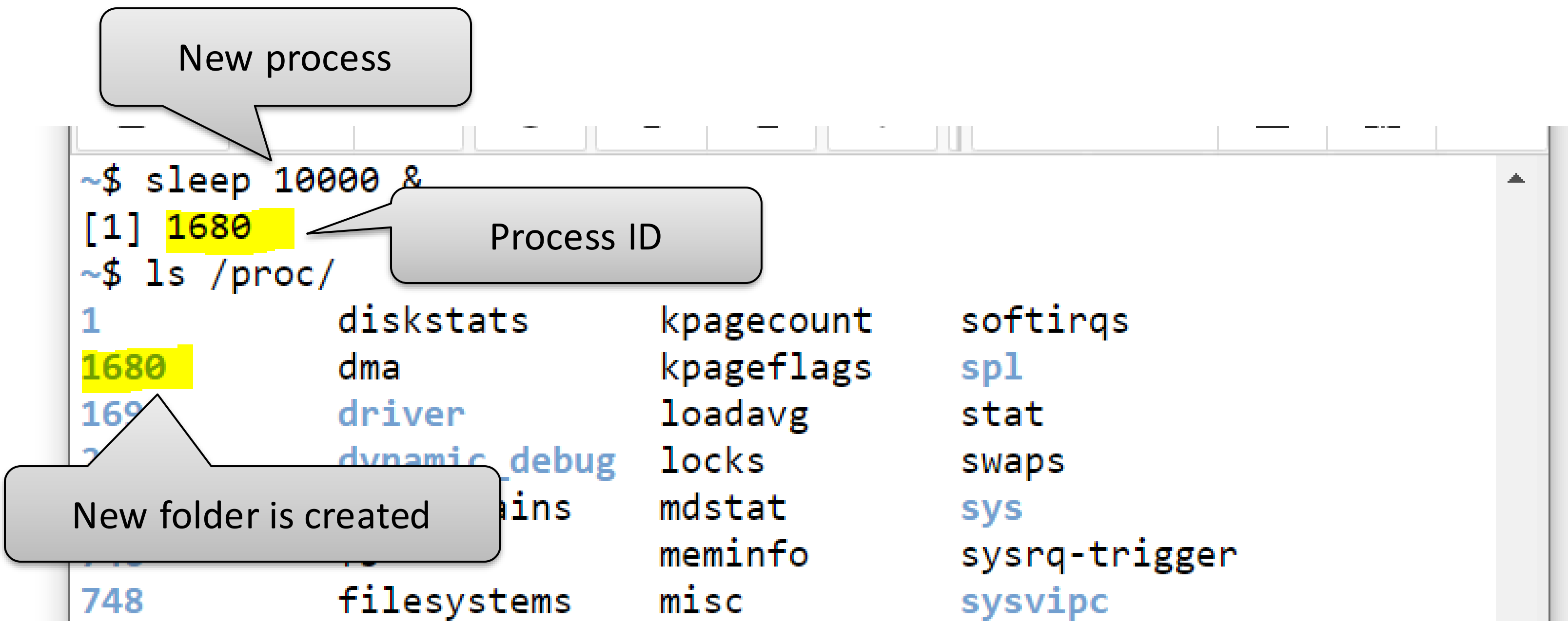
The /proc/ directory contains a wealth of information detailing system hardware and any running processes. In addition, some of the files within /proc/ can be manipulated by users and applications to communicate configuration changes to the kernel([fedora documentation]()).



Linux

proc - process information pseudo-filesystem

Every process in Linux is represented by a directory in /proc e.g.



Linux

proc - process information pseudo-filesystem

```
~$ sleep 10000 &
[1] 1680
```

```
~$ ls /proc/1680/
arch_status         fd          numa_maps      smaps_rollup
attr                fdinfo      oom_adj        stack
autogroup           gid_map     oom_score      stat
auxv                io          oom_score_adj  statm
cgroup              limits      pagemap        status
clear_refs          loginuid    patch_state    syscall
cmdline             map_files   personality    task
comm                maps        projid_map     timens_offsets
coredump_filter     mem         root           timers
cpu_resctrl_groups  mountinfo   sched          timerslack_ns
cpuset              mounts      schedstat      uid_map
cwd                 mountstats  sessionid      wchan
environ             net         setgroups
exe                 ns          smaps
~$
```

Linux

proc - process information pseudo-filesystem

e.g. process address space

```
~$ sleep 10000 &
[1] 1680
```

```
~$ cat  /proc/1680/maps
5558f1db2000-5558f1db4000 r--p 00000000 00:cf 26481965   /usr/bin/sleep
5558f1db4000-5558f1db8000 r-xp 00002000 00:cf 26481965   /usr/bin/sleep
5558f1db8000-5558f1dba000 r--p 00006000 00:cf 26481965   /usr/bin/sleep
5558f1dbb000-5558f1dbc000 r--p 00008000 00:cf 26481965   /usr/bin/sleep
5558f1dbc000-5558f1dbd000 rw-p 00009000 00:cf 26481965   /usr/bin/sleep
5558f326d000-5558f328e000 rw-p 00000000 00:00 0          [heap]
7f31af675000-7f31af6a7000 r--p 00000000 00:cf 41908341   /usr/lib/locale/C.UTF-8/LC_CTYPE
7f31af6a7000-7f31af81a000 r--p 00000000 00:cf 41908340   /usr/lib/locale/C.UTF-8/LC_COLLATE
7f31af81a000-7f31af83c000 r--p 00000000 00:cf 41899982   /lib/x86_64-linux-gnu/libc-2.31.so
7f31af83c000-7f31af9b4000 r-xp 00022000 00:cf 41899982   /lib/x86_64-linux-gnu/libc-2.31.so
7f31af9b4000-7f31afa02000 r--p 0019a000 00:cf 41899982   /lib/x86_64-linux-gnu/libc-2.31.so
7f31afa02000-7f31afa06000 r--p 001e7000 00:cf 41899982   /lib/x86_64-linux-gnu/libc-2.31.so
7f31afa06000-7f31afa08000 rw-p 001eb000 00:cf 41899982   /lib/x86_64-linux-gnu/libc-2.31.so
7f31afa08000-7f31afa0e000 rw-p 00000000 00:00 0
7f31afa3a000-7f31afa3b000 r--p 00000000 00:cf 41908346   /usr/lib/locale/C.UTF-8/LC_NUMERIC
7f31afa3b000-7f31afa3c000 r--p 00000000 00:cf 41908349   /usr/lib/locale/C.UTF-8/LC_TIME
```

Linux

process monitoring tools

Pstree, ps, and top(htop)

```
pstree - display a tree of processes
e.g.
        pstree –p -T

ps - report a snapshot of the current processes
e.g.
        ps aux

top - display Linux processes
e.g.
        top
```

Linux

process monitoring tools

Pstree, ps, and top(htop)

```
~$
~$ pstree -p -T
tini(1)───sh(7)───node(8)─┬─bash(746)───pstree(859)
                          ├─bash(748)
                          └─sshd(20)
~$
```

```
~$
~$ ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:00 /cocalc/bin/tini -v -g -- sh -c env -i
    7 ?        SN     0:00 sh -c env -i /cocalc/init/init.sh $COC
    8 ?        RNl    0:10 node --optimize-for-size --gc-interval
   20 ?        SN     0:00 sshd: /usr/sbin/sshd -D -p 2222 -h /tr
  314 pts/0    SNs    0:00 /bin/bash
  401 pts/0    SN     0:00 sleep 10000
  403 pts/0    RN+    0:00 ps ax
~$
~$
```

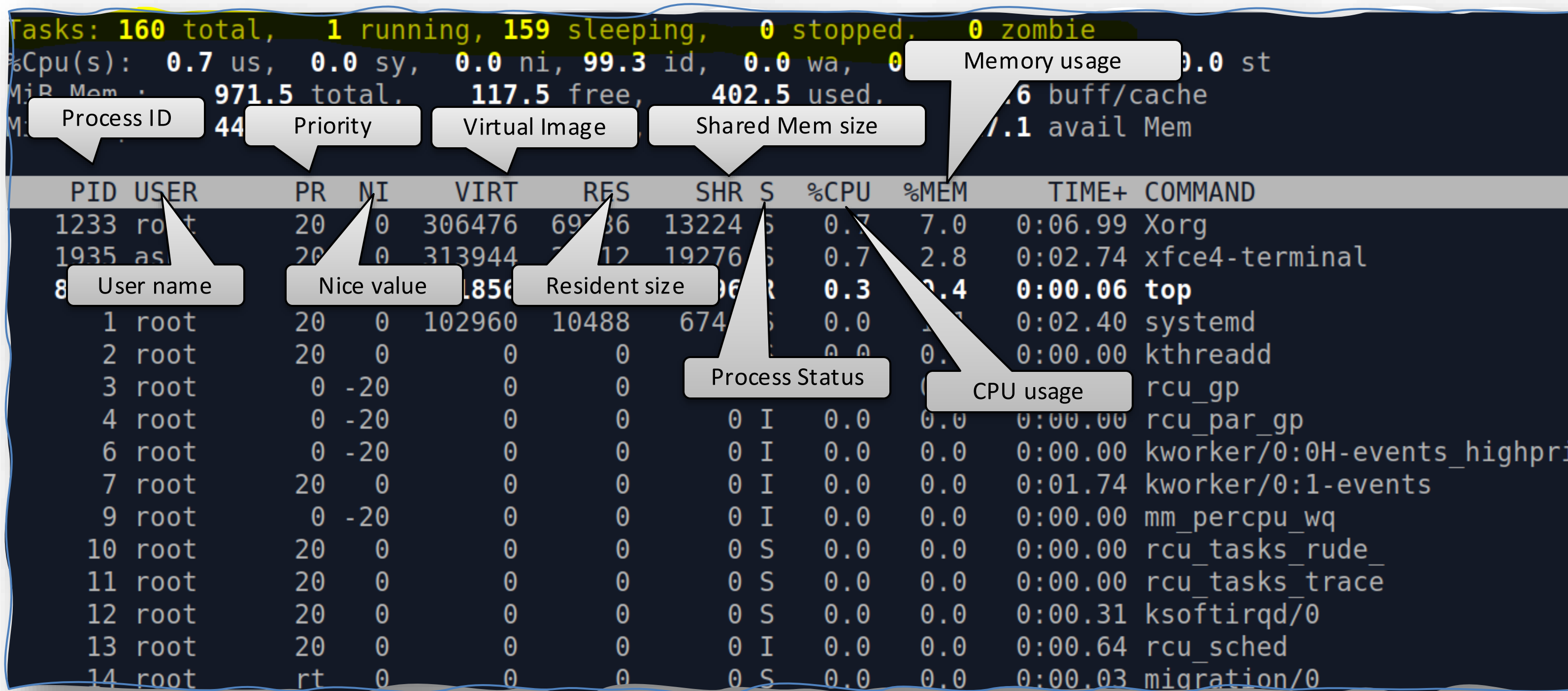
Linux

process monitoring tools

Pstree, ps, and **top**(htop)

```
Tasks: 160 total,   1 running, 159 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.7 us,  0.0 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :    971.5 total,    117.5 free,    402.5 used,    451.6 buff/cache
MiB Swap:    448.5 total,    307.1 free,    141.4 used.    417.1 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 1233 root      20   0  306476  69736  13224 S   0.7   7.0   0:06.99 Xorg
 1935 asp       20   0  313944  27712  19276 S   0.7   2.8   0:02.74 xfce4-terminal
 8140 asp       20   0   11856   3604   3096 R   0.3   0.4   0:00.06 top
    1 root      20   0  102960  10488   6744 S   0.0   1.1   0:02.40 systemd
    2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd
    3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
    6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-events_highpri
    7 root      20   0       0      0      0 I   0.0   0.0   0:01.74 kworker/0:1-events
    9 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
   10 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_tasks_rude_
   11 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_tasks_trace
   12 root      20   0       0      0      0 S   0.0   0.0   0:00.31 ksoftirqd/0
   13 root      20   0       0      0      0 I   0.0   0.0   0:00.64 rcu_sched
   14 root      rt   0       0      0      0 S   0.0   0.0   0:00.03 migration/0
```

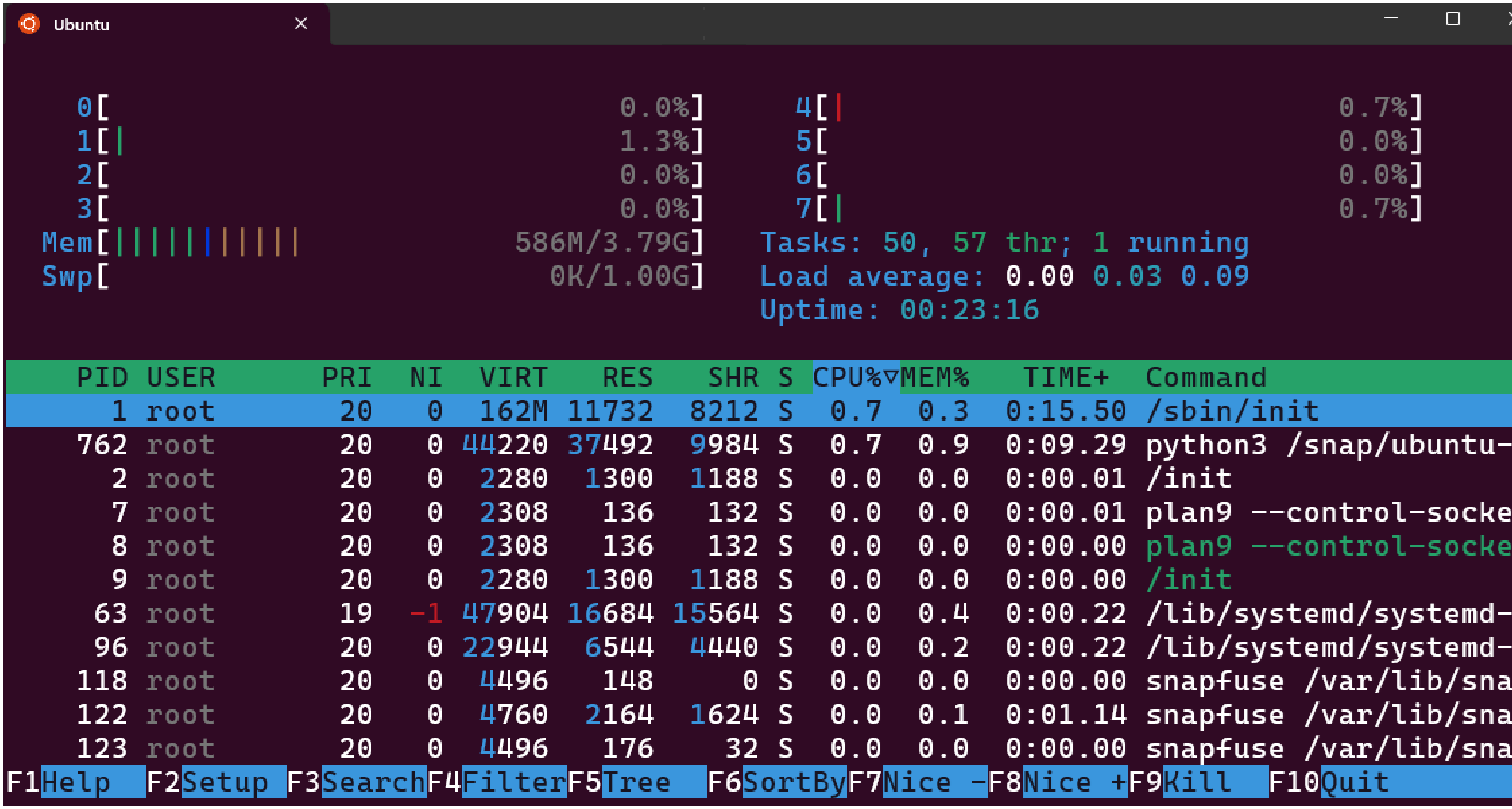
Linux
process monitoring tools
Pstree, ps, and **top**(htop)



Linux

process monitoring tools

Pstree, ps, and top(**htop**)



Linux

Process state

man ps

```
D  uninterruptible sleep (usually IO and cannot be killed)
I  Idle kernel thread
R  running or runnable (on run queue)
S  interruptible sleep (waiting for an event to complete and can
   be killed)
T  stopped by job control signal
X  dead (should never be seen)
Z  defunct ("zombie") process, terminated but not reaped by its
   parent

<  high-priority (not nice to other users)
N  low-priority (nice to other users)
+  is in the foreground process group
```

Linux

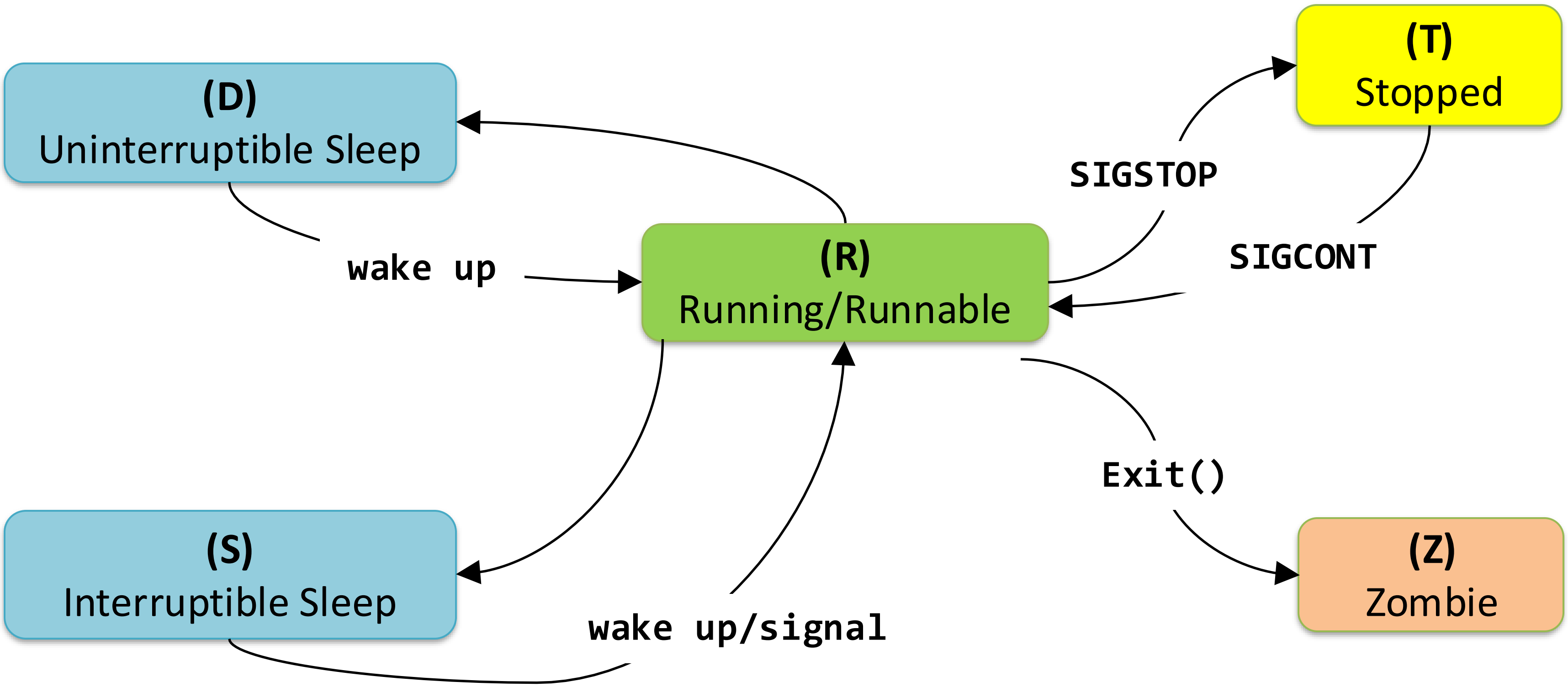
Process state

S interruptible sleep (waiting for an event to complete)

```
~$
~$ ps ax
    PID TTY      STAT   TIME COMMAND
      1 ?        Ss     0:00 /cocalc/bin/tini -v -g -- sh -c env -i
      7 ?        SN     0:00 sh -c env -i /cocalc/init/init.sh $COC
      8 ?        RNl    0:10 node --optimize-for-size --gc-interval
     20 ?        SN     0:00 sshd: /usr/sbin/sshd -D -p 2222 -h /tm
    314 pts/0    SNs    0:00 /bin/bash
    401 pts/0    SN     0:00 sleep 10000
    403 pts/0    RN+    0:00 ps ax
~$
```

Linux

Process state



Linux

Process state

```
 T    stopped by job control signal
```

```
asp@asp-VirtualBox:~/Desktop/OS$ ./a.out
[1]+  Stopped                 ./a.out
asp@asp-VirtualBox:~/Desktop/OS$
```

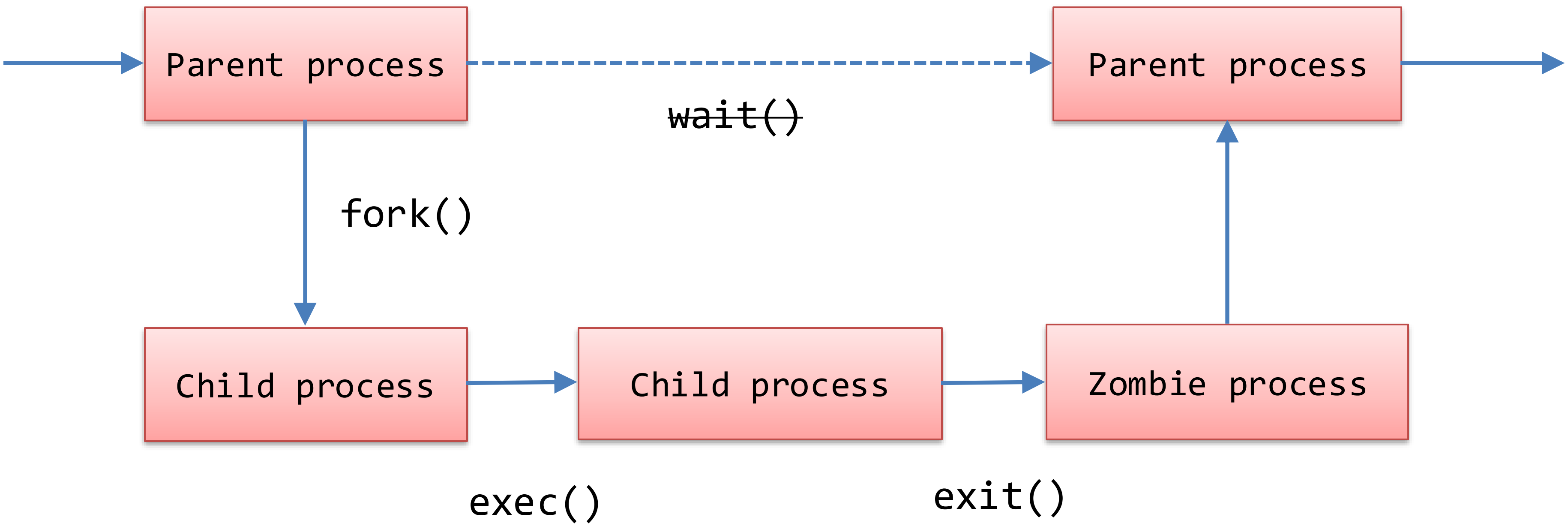
```
Tasks: 161 total,   1 running, 159 sleeping,   1 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.0 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :    971.5 total,     87.0 free,    446.5 used,    438.1 buff/cache
MiB Swap:    448.5 total,    307.4 free,    141.1 used.    367.3 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   8266 asp       20   0    2364    512    448 T   0.0   0.1   0:00.00 a.out
```

Linux

Zombie process

The parent process is responsible for waiting for child process to terminate and then cleaning up the child process entry from the process table.

The zombie process can be easily created when the parent process does not wait for the child process termination. A zombie process doesn't consume many system resources, but it does continue to live in the system's task list.



Linux

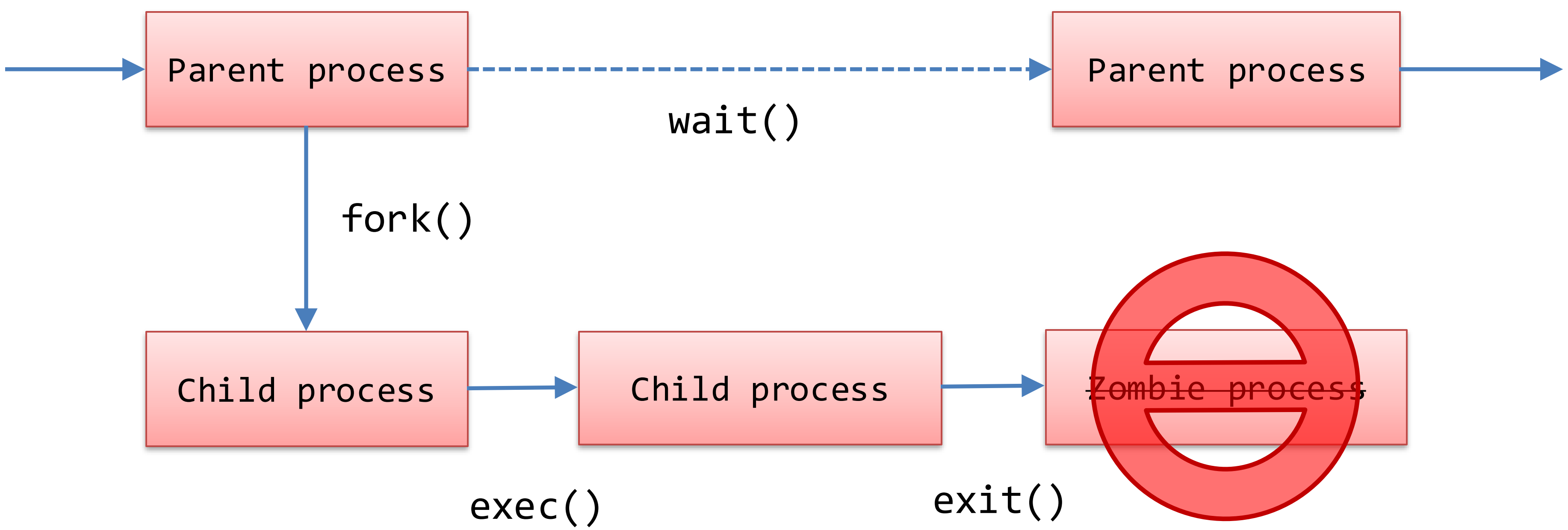
Zombie process

```
~$ ./a.out
Child: pid is 1550
Parent: pid is 1549
~$
```

```
top - 23:58:38 up  4:15,  0 users,  load average: 1.84, 1.99, 2.58
Tasks:   9 total,   1 running,   7 sleeping,   0 stopped,   1 zombie
%Cpu(s): 13.2 us,  6.0 sy,  1.8 ni, 75.4 id,  2.7 wa,  0.0 hi,  0.8 si,  0.0 st
MiB Mem :  32104.4 total,  10043.8 free,   3500.7 used,  18559.8 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.  26939.4 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    8 user      38  18  963500 111260  33800 S   2.0   0.3   1:27.32 node
    1 user      20   0    2500    584    520 S   0.0   0.0   0:00.06 tini
    7 user      38  18    2616    600    528 S   0.0   0.0   0:00.00 sh
   20 user      38  18   12180   6788   5956 S   0.0   0.0   0:00.01 sshd
  363 user      38  18    8116   6056   3308 S   0.0   0.0   0:00.13 bash
  910 user      38  18    8116   6232   3484 S   0.0   0.0   0:00.08 bash
 1549 user      38  18    2364    512    444 S   0.0   0.0   0:00.00 a.out
 1550 user      38  18       0      0      0 Z   0.0   0.0   0:00.00 a.out
 1551 user      38  18    7904   3612   3096 R   0.0   0.0   0:00.00 top
```

Linux

Zombie process



Linux

Zombie process

```
~$ ./a.out
Child: pid is 1850
Parent: pid is 1849
~$
```

```
top - 00:14:44 up  4:31,  0 users,  load average: 1.55, 1.86, 2.08
Tasks:   8 total,   1 running,   7 sleeping,   0 stopped,   0 zombie
%Cpu(s): 11.3 us,  4.4 sy,  1.8 ni, 81.3 id,  0.7 wa,  0.0 hi,  0.6 si,  0.0 st
MiB Mem :  32104.4 total,  10316.8 free,   3168.3 used,  18619.2 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.  27271.3 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
8	user	38	18	964264	113720	33800	S	10.6	0.3	2:11.88	node
1	user	20	0	2500	584	520	S	0.0	0.0	0:00.08	tini
7	user	38	18	2616	600	528	S	0.0	0.0	0:00.00	sh
20	user	38	18	12180	6788	5956	S	0.0	0.0	0:00.01	sshd
363	user	38	18	8116	6056	3308	S	0.0	0.0	0:00.14	bash
910	user	38	18	8116	6232	3484	S	0.0	0.0	0:00.08	bash
1848	**user**	**38**	**18**	**7904**	**3540**	**3024**	**R**	**0.0**	**0.0**	**0:00.00**	**top**
1849	user	38	18	2364	572	508	S	0.0	0.0	0:00.00	a.out

Linux

Orphan process

The parent dies (or terminates) before its child. It means that the child process is still running even thought the parent process has terminated. The child process will be automatically adopted by the **init** process (the parent of all the processes with the pid of 1).

```
~$ pstree -p -T
tini(1)───sh(6)───node(7)─┬─bash(273)───pstree(476)
                          ├─bash(433)───a.out(474)───a.out(475)
                          └─sshd(19)
~$ pstree -p -T
tini(1)─┬─a.out(475)
        └─sh(6)───node(7)─┬─bash(273)───pstree(477)
                          ├─bash(433)
                          └─sshd(19)

~$
```

Linux

Orphan process