processes
process

- a program in execution (running) on a computer
- characterized by...
  - at least one execution thread
  - an associated set of system resources
  - a current state of CPU (and possibly other resources)
threads

• the entity that can be assigned to, and executed on, a processor
  – it is meaningful only within a process
  – described by
    • the value of the program counter
    • the value of the CPU registers

• in modern operating systems a process may contains one or more thread

• we assume it contains one thread
  – unless otherwise specified
summary

• OS – process interaction
  – the point of view of the process
  – the point of view of the OS

• system calls

• process lifecycle

• state diagrams for processes

• representation within OS
the point of view of the process

- it explicitly interacts with OS by means of system calls (syscalls)
- like procedure calls but...
  - syscall are made available by OS
  - can perform privileged operations
- syscalls are not called using regular “call” instructions
  - special instruction
  - software interrupt
OS and processes interaction

executes in user mode

executes in kernel mode

Kernel
classes of syscalls

- **I/O**
  - read and write (need a communication channel)

- **resources allocation/deallocation**
  - communication channels (with i/o devices or other processes)
  - memory
  - etc.

- **processes control**
  - create, kill, wait for..., stop, continue, debug, etc.

- **resource management (i.e. miscellanea)**
  - change attributes (for files, devices, comm. channels, etc.)
  - set system values (sys. clock, routing table, ecc.)

*not on the book*
the point of view of the OS

• when a syscall is executed, OS can...
  – … immediately doing what it is asked for and returning to process execution
  – … postpone the request and blocking the process until the request can be fulfilled
    • e.g. a read syscall may require a disk operation, so read cannot be immediately fulfilled
the point of view of the OS

- often more processes can be executed
  - but cpu is only one (or are limited in number)
- OS can interleave the execution of multiple processes
- OS can choose which one to run
  - maximize processor utilization
  - providing reasonable response time
- decisions are taken by the “short time cpu scheduler”
a model of process lifecycle

- **creation**
  - why and how a process is created?

- **execution**
  - regular “unprivileged” computation
  - syscalls (possibly blocking)

- **termination**
  - regular (it asks the OS to terminate)
  - error (illegal instruction, div. by zero, ecc.)
## Process Creation

### Table 3.1 Reasons for Process Creation

<table>
<thead>
<tr>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New batch job</td>
<td>The operating system is provided with a batch job control stream, usually on tape or disk. When the operating system is prepared to take on new work, it will read the next sequence of job control commands.</td>
</tr>
<tr>
<td>Interactive logon</td>
<td>A user at a terminal logs on to the system.</td>
</tr>
<tr>
<td>Created by OS to provide a service</td>
<td>The operating system can create a process to perform a function on behalf of a user program, without the user having to wait (e.g., a process to control printing).</td>
</tr>
<tr>
<td>Spawned by existing process</td>
<td>For purposes of modularity or to exploit parallelism, a user program can dictate the creation of a number of processes.</td>
</tr>
</tbody>
</table>
process creation

• but...

• in modern operating systems all causes are implemented by

  process spawning

• but for the first process!
  – it is created at boot time and never dies (usually)
processes tree

• every process has a parent
  – the one that asked for its creation
• but for the first process
  – which is the root of the tree
## Process Termination

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal completion</td>
<td>The process executes an OS service call to indicate that it has completed running.</td>
</tr>
<tr>
<td>Time limit exceeded</td>
<td>The process has run longer than the specified total time limit. There are a number of possibilities for the type of time that is measured. These include total elapsed time (&quot;wall clock time&quot;), amount of time spent executing, and, in the case of an interactive process, the amount of time since the user last provided any input.</td>
</tr>
<tr>
<td>Memory unavailable</td>
<td>The process requires more memory than the system can provide.</td>
</tr>
<tr>
<td>Bounds violation</td>
<td>The process tries to access a memory location that it is not allowed to access.</td>
</tr>
<tr>
<td>Protection error</td>
<td>The process attempts to use a resource such as a file that it is not allowed to use, or it tries to use it in an improper fashion, such as writing to a read-only file.</td>
</tr>
<tr>
<td>Arithmetic error</td>
<td>The process tries a prohibited computation, such as division by zero, or tries to store numbers larger than the hardware can accommodate.</td>
</tr>
</tbody>
</table>
## Process Termination

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time overrun</td>
<td>The process has waited longer than a specified maximum for a certain event to occur.</td>
</tr>
<tr>
<td>I/O failure</td>
<td>An error occurs during input or output, such as inability to find a file, failure to read or write after a specified maximum number of tries (when, for example, a defective area is encountered on a tape), or invalid operation (such as reading from the line printer).</td>
</tr>
<tr>
<td>Invalid instruction</td>
<td>The process attempts to execute a nonexistent instruction (often a result of branching into a data area and attempting to execute the data).</td>
</tr>
<tr>
<td>Privileged instruction</td>
<td>The process attempts to use an instruction reserved for the operating system.</td>
</tr>
<tr>
<td>Data misuse</td>
<td>A piece of data is of the wrong type or is not initialized.</td>
</tr>
<tr>
<td>Operator or OS intervention</td>
<td>For some reason, the operator or the operating system has terminated the process (for example, if a deadlock exists).</td>
</tr>
<tr>
<td>Parent termination</td>
<td>When a parent terminates, the operating system may automatically terminate all of the offspring of that parent.</td>
</tr>
<tr>
<td>Parent request</td>
<td>A parent process typically has the authority to terminate any of its offspring.</td>
</tr>
</tbody>
</table>
process termination

- normal completion is asked by a process by calling a specific syscall
- other form of termination when something wrong is detected...
  - during the execution of a syscall
    - of the process (e.g. memory unavailable)
    - of other processes (e.g. regular termination of A implies killing child B)
  - during handling of an interrupt
    - e.g. div. by zero, illegal instruction, illegal memory access, etc.
scheduling and dispatching

- **scheduling**
  - deciding which is the next process executed by the CPU

- **dispatching**
  - setting up CPU registers to execute the process
    - i.e. restore the context for the process
scheduler vs. dispatcher

- scheduling and dispatching are usually performed together by the same routine
- we use “scheduler” or “dispatcher” depending on the aspect we need to emphasize
dispatching example

• what instructions are executed by the cpu?
Processes and Memory

![Diagram showing processes and memory allocation]

Main Memory

- Address 0
- 100: Dispatcher
- 5000: Process A
- 8000: Process B
- 12000: Process C

Program Counter: 8000
Trace of Process

- Sequence of instruction (addresses) for each process

<table>
<thead>
<tr>
<th></th>
<th>5000</th>
<th>5001</th>
<th>5002</th>
<th>5003</th>
<th>5004</th>
<th>5005</th>
<th>5006</th>
<th>5007</th>
<th>5008</th>
<th>5009</th>
<th>5010</th>
<th>5011</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process A</td>
<td>8000</td>
<td>8001</td>
<td>8002</td>
<td>8003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process C</td>
<td>12000</td>
<td>12001</td>
<td>12002</td>
<td>12003</td>
<td>12004</td>
<td>12005</td>
<td>12006</td>
<td>12007</td>
<td>12008</td>
<td>12009</td>
<td>12010</td>
<td>12011</td>
</tr>
</tbody>
</table>

5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C
Dispatcher

- The *dispatcher* switches the processor from one process to another *(process switch)*.
Two-State Process Model

(a) State transition diagram

(b) Queuing diagram
Five-State Process Model

Figure 3.6 Five-State Process Model
Process States

Figure 3.7  Process States for Trace of Figure 3.4
One sequential I/O device
Many sequential I/O devices

(b) Multiple blocked queues
Suspended Processes

• Processor is faster than I/O so many processes could be waiting for I/O
• Swap these processes to disk to free up memory
• Blocked state becomes suspend state when swapped to disk
• Two new states
  – Blocked/Suspend
  – Ready/Suspend
Two New States

(b) With Two Suspend States
# Several Reasons for Process Suspension

<table>
<thead>
<tr>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swapping</td>
<td>The operating system needs to release sufficient main memory to bring in a process that is ready to execute.</td>
</tr>
<tr>
<td>Other OS reason</td>
<td>The operating system may suspend a background or utility process or a process that is suspected of causing a problem.</td>
</tr>
<tr>
<td>Interactive user request</td>
<td>A user may wish to suspend execution of a program for purposes of debugging or in connection with the use of a resource.</td>
</tr>
<tr>
<td>Timing</td>
<td>A process may be executed periodically (e.g., an accounting or system monitoring process) and may be suspended while waiting for the next time interval.</td>
</tr>
<tr>
<td>Parent process request</td>
<td>A parent process may wish to suspend execution of a descendent to examine or modify the suspended process, or to coordinate the activity of various descendents.</td>
</tr>
</tbody>
</table>
process description
Table 3.4 Typical Elements of a Process Image

| User Data       | The modifiable part of the user space. May include program data, a user stack area, and programs that may be modified. |
| User Program    | The program to be executed. |
| System Stack    | Each process has one or more last-in-first-out (LIFO) system stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls. |
| Process Control Block | Data needed by the operating system to control the process (see Table 3.5). |
OS controls assignment of resources to processes
Process Control Block (PCB)

- contains data about **one** process
  - one instance for each process
- contains all the information we need to...
  - ...interrupt a running process
  - ...resume execution
- created and managed by the operating system
- allows support for multiple processes

<table>
<thead>
<tr>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
</tr>
<tr>
<td>Priority</td>
</tr>
<tr>
<td>Program counter</td>
</tr>
<tr>
<td>Memory pointers</td>
</tr>
<tr>
<td>Context data</td>
</tr>
<tr>
<td>I/O status information</td>
</tr>
<tr>
<td>Accounting information</td>
</tr>
</tbody>
</table>

-
Process Elements in PCB

*they largely depend on the OS*

- Process Identifier (PID)
- State (ready, blocked, etc.)
  - if blocked, events the process is waiting for
- Priority (for the scheduler)
- saved CPU registers and PC (a.k.a. context)
- Memory pointers (program, data, stack, tables, etc.)
- I/O status information (open files, outstanding I/O requests, inter-processes communication, etc)
- Accounting information (CPU time used, limits, etc.)
- user that owns the process, and/or privileges
- process that created the process
Data Structuring

- PCB – PCB pointers
  - parent-child (creator-created) relationship with another process

- queues
  - all processes in a waiting state for a particular priority level may be linked in a queue.
Process Creation

• Assign a unique process identifier
• Allocate space for the process
• Initialize process control block
• Set up appropriate linkages
  – e.g. add new process to linked list used for scheduling queue
• Create or expand other data structures
  – e.g. maintain an accounting file
PCB synonyms

- process descriptor
- task control block
- task descriptor

linux

- task_struct
PCB related data structures

- process table
- memory tables
- I/O tables
- file tables
Process Table

• one entry for each process
• contains a minimal amount of information needed to activate the process
  – usually a “pointer” to the PCB
  – it may be a complex data structure (tree, hash table, ecc.)
Memory Tables

• Allocation of main memory to processes
• Allocation of secondary memory to processes
• Protection attributes for access to shared memory regions
• Information needed to manage virtual memory
I/O Tables

- I/O device is available or assigned
- Status of I/O operation
- Location in main memory being used as the source or destination of the I/O transfer
File Tables

- Existence of files
- Location on secondary memory
- Current Status
- Attributes
- Sometimes this information is maintained by a file management system
process control
mode switch

- **two cases**
  - **user-mode → kernel-mode**
    - triggered by an interrupt or a system call
    - set cpu in privileged mode
    - may save the cpu state
  - **kernel-mode → user-mode**
    - triggered by the kernel when it “decides” to resume process execution
    - set cpu in unprivileged mode
    - may restore all or part of the cpu state
process switch (dispatching)

- a process switch assigns the CPU to a different process
  - before: \( P_1 \) running, \( P_2 \) ready
  - after: \( P_1 \) not running, \( P_2 \) running

- it is performed in kernel-mode
  - it requires two mode switches
    1. user-mode \( \rightarrow \) kernel-mode before the process switch
       - triggered by interrupt, trap or system call
       - kernel possibly fulfill a request (e.g. I/O)
    2. kernel-mode \( \rightarrow \) user-mode after the process switch
       - into the process chosen by the kernel (scheduler)
process switch

- it modifies OS data structures
  - set proper state in PCB of $P_1$ and $P_2$
  - update queues
    - move $P_1$ into the appropriate queue
    - move $P_2$ out of the ready queue
  - update CPU memory tables for the image of $P_2$

- the next mode switch (kernel-mode $\rightarrow$ user-mode) will restore the cpu state of $P_2$
typical situations for switching mode and/or process

• **clock interrupt**
  - process has executed for the maximum allowable time slice
  - always switch process

• **system call**
  - process switch when it is a blocking I/O request
  - OS may check if other processes have greater priority and possibly switch process
typical situations for switching mode and maybe process

- I/O interrupt
  - a blocked process may become ready
  - process switch depends on OS policies and priorities

- other interrupts (a.k.a traps)
  - memory page fault (virtual memory)
    - current process becomes blocked (waiting for the page) and process is switched
  - error or exception
    - current process usually die and process is switched
execution of the OS

- the OS is executed by the cpu
memory: processes vs. OS

- MMU configuration is critical during mode/process switch
- OS needs to access memory of the processes
- OS need to access its own data/code
- several memory layout approaches are possible
  - non-process kernel
  - kernel execution within user processes
  - process-based operating system
“non-process kernel”

- kernel has its own “memory space”
- memory space switched at each mode switch
  - inefficient
    - memory cache should be flushed
    - kernel must implements tricks to access the images of processes
- useful when kernel code and data are big w.r.t. physical memory size
  - obsolete
“Execution Within User Processes” (a.k.a. monolithic)

- also called “monolithic”
- the whole kernel (data and code) appears in the memory layout of each process
  - shared pages

Proc 1

Proc 2

Proc 3

kernel

kernel

kernel
Execution Within User Processes

- Each process has its own **image**
- Image contains also
  - Kernel stack
  - Kernel program
  - Kernel data

- Kernel program and data are shared by all images
  - Kernel mode is needed to read and write them
“Execution Within User Processes”

• no MMU reconfiguration is needed during a mode switch
  – efficient

• kernel can access current processes memory without any trick

• waste of virtual address space for the kernel
Execution Within User Processes

- to fulfill a system call or interrupt...
  - mode is switched
  - current memory image remain the same
  - both kernel data and current process data can be accessed
- a MMU reconfiguration occurs only when a new process is dispatched
process-based OS (microkernel)

• like “execution in user process” but kernel functionalities are minimal
  – thread/process scheduling and dispatching
  – Inter Process Communication (IPC)
  – direct access by os-processes to hardware

• implements many OS functionalities as a system process
  – process switch and inter-process communication
  – many system calls are actually IPC messages
process-based OS (microkernel)

• modular and robust
  – code needing kernel mode is small
  – services may be added, removed or distributed

• usually less efficient than “kernel execution within user process”
process-based OS: design choices

- may OS-processes run in kernel mode to access hardware?
- drivers are implemented in the kernel or as processes?
- consider the efficiency of the alternatives of an I/O operation
  - how many inter-processes messages?
  - how many mode switches?
  - how many process switches?
  - how many times dispatcher should run to execute an I/O?
real life OS

- unix BSD – monolithic “exec within user proc.”
- Windows starts as microkernel, currently hybrid
- Linux starts as monolithic, currently hybrid
  - system is “executed within user process”
  - some OS tasks are demanded to special processes (kernel threads)
  - modular, efficient, not reliable as microkernel
- pure microkernels
  - mach, chorus, L4, minix
  - real time: QNX, VxWorks
- Mac OSX – hybrid (mach + BSD)
real life OS: windows

- MS started with microkernel in mind
- not real a microkernel since NT4
  - e.g. kernel contains graphic code