Process Management: Processes and Threads

CS 342 – Operating Systems

Ibrahim Korpeoglu
Bilkent University
Computer Engineering Department

What is a process?

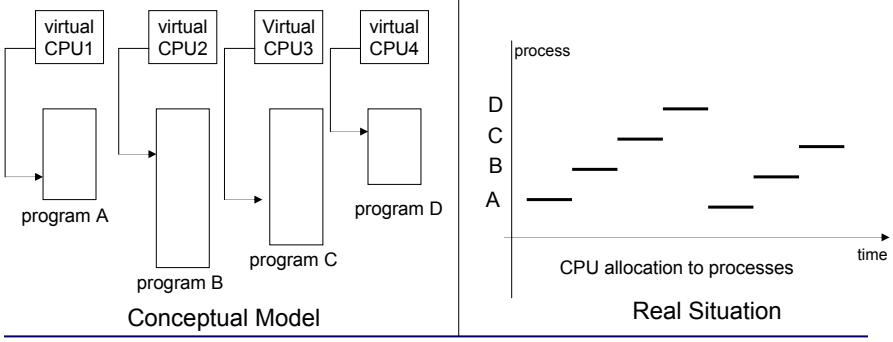
- One of the fundamental functions of an OS is processes management
- A process is (in simple terms) a program in execution
- A **program** (an executable file) is a *passive* entity that resides on a disk (floppy, cd-rom, hard-disk)
- A **process** is an *active* entity that can run. It includes
  - The program (executable file) loaded in memory (called *memory (or core) image* of the process)
  - The state: set of register values in CPU (PC, SP, other registers)
  - Open files: their descriptors and their status.
  - Allocated I/O resource for this process,
  - ….

What is a process

- Modern computer can run multiple processes using a single CPU
- A scheduler coordinates the usage of CPU by several processes
- A user that is using a process perceives usually that the CPU is allocated to himself/herself.
- This is also pseudo-parallelism, where more than one process run in parallel.
- At any given time instant, only one process is running: using the CPU

The Process Model

- Conceptually, each process has its own virtual CPU: virtual registers.
- In reality, a single physical CPU is shared.

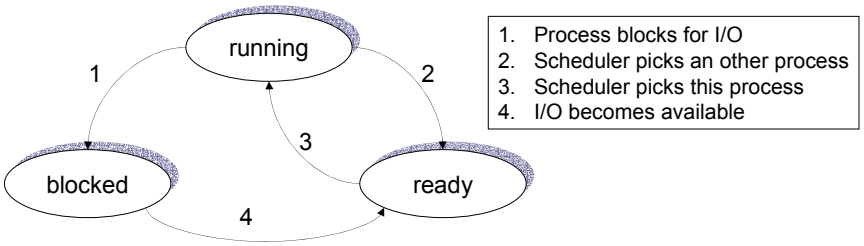


What is a process

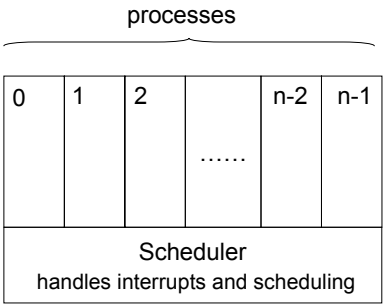
- When a process is created, the respective program is loaded into memory from disk
- A program consists of 3 parts (segments)
  - Text segment: includes sequence of instructions
    - Never modified by the process
  - Data segment: contains global variables
    - Can be modified by the process during execution
  - Stack segment: holds temporary variables
    - Local variable of functions
    - Function parameters
    - Return addresses

Process State

- In a multiprograming environment, a process changes state while in execution.
- The states that a process can be in are:
  - Running
  - Ready
  - Blocked (waiting)



Process-structured OS



A process can be a system process doing for example a file service

Process Implementation

- A process is represented in OS using a structure called *process control block* (PCB)
- A PCB is entry in a table, called *process table*
- When a process is suspended (changes from running state to one of blocked or ready states)
  - All its state information is stored in the corresponding PCB in process table
- When a process is resumed (changed from blocked or ready state to running state)
  - All its state information contained in PCB is loaded into CPU.
  - The process can continue executing.

What is in PCB?

- A PCB contains information about a process classified as follows
  - Process state: ready, blocked, running.
  - Program counter: address of the next instruction that is to be executed for this process
  - CPU registers: vary depending on hardware
    - Accumulators
    - Index registers
    - Stack pointers
    - General purpose registers
    - Condition code information
  - CPU scheduling information
    - Process priority
    - Pointers to scheduling queues.
  - Memory Management Information
    - Base and limit register values
    - Page tables
    - Text, data and stack pointers, etc.
  - Accounting Information
    - Amount of CPU usage, time limits
    - Process ID
  - I/O status information
    - List of I/O devices allocated to the process
    - List of open files, etc.

A typical PCB

Process Management	Memory Management	File Management
Registers	Pointer to text segment	Root directory
Program Counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	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		

The above table contains come of the fields that may exist in a PCB. The list is not complete.

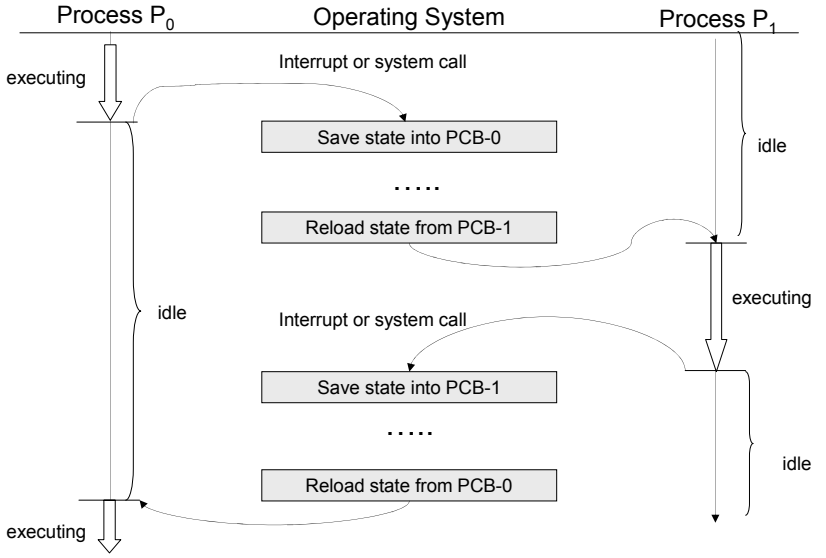
How interrupts affect processes?

- A hardware interrupt may come to the CPU requesting some immediate service from CPU.
- CPU may be running some process at that time.
- CPU should resume process and should handle the interrupt.

How interrupts affect processes?

- Steps to handle interrupts
  - Hardware stacks program counter, psw, and possible one more other registers.
  - Hardware loads new program counter from interrupt vector
    - Interrupt vector keeps the address of ISR (interrupt service routine) for the incoming interrupt identified by a number.
  - In ISR:
    - An assembly language procedure saves the registers into the PCB of the process that was executing.
    - An assembly language procedure sets up a new stack
      - Information stored on the previous stack is removed
    - Interrupt service runs (a C program)
    - Scheduler decides which process is run next.
    - C procedure returns to the assembly code
    - Assembly language procedure starts up new current process

Context switch



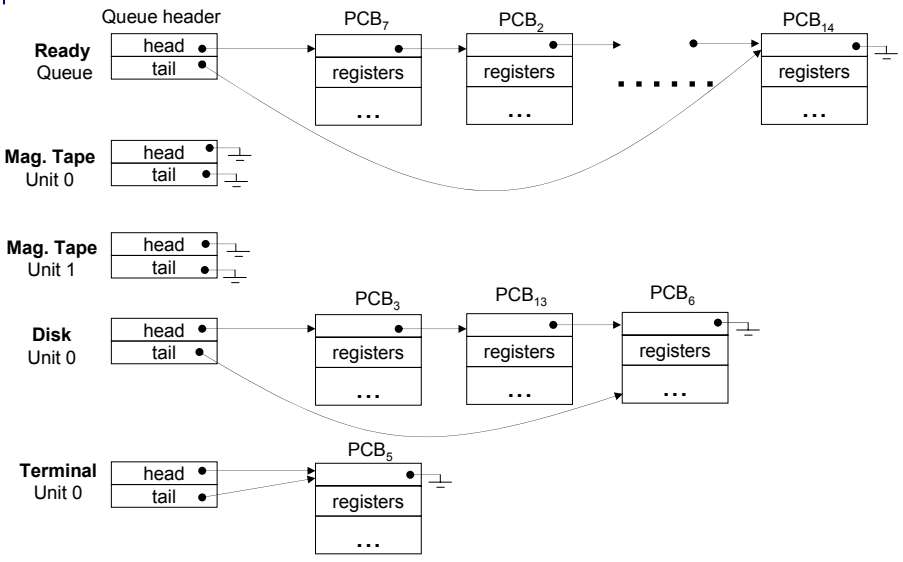
Context switch

- Switching the CPU to an other process requires saving the state of the old process and loading the saved state of the new process into CPU. This is called *context switch*
- The context of a process is represented in the PCB of that process
- System does not do any useful work during context switch. Hence it is waste of time.
- Context switch is usually costly.
  - Context switch time may range from 1μs to 1000μs

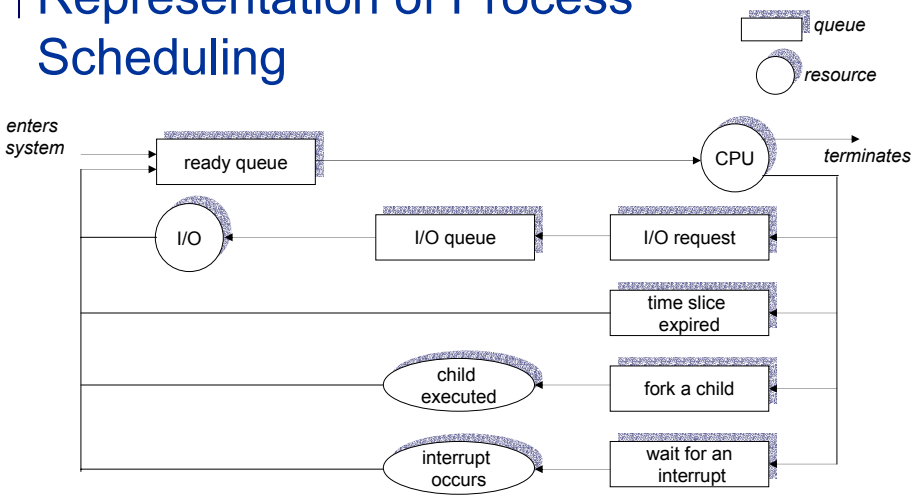
Process Scheduling

- There is scheduler component of OS that decides which process should use CPU and when, and how long.
- When a process enters the system, it is put into a *process (job) queue.*
  - *This queue constists of all processes in the system*
- All processes in the ready state are put in *ready queue*.
  - A linked list.
- OS has also other queues:
  - Each I/O device may have a queue associated with it. This queue contains all the processes that want service from that device. This is called *device queue*.
  - Each device has its own device queue.

Various Queues in an OS



Representation of Process Scheduling



Queuing-diagram representation of process scheduling

Schedulers

- A process goes through various scheduling queues during its lifetime.
- OS must select processes from these queues for serving them
- The selection process is carried out the appropriate scheduler.

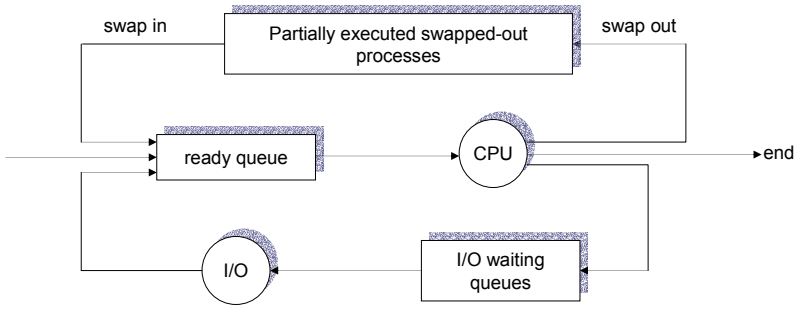
Schedulers

- There are 3 kinds of schedulers
  - Long-term scheduler
  - Short-term scheduler
  - Medium-term scheduler
  - Invocation intervals of these schedulers differ.
- Long-term scheduler:
  - For batch systems, selects the set of jobs that needs to be loaded into memory and executed until they finish.
  - When a terminated process leaves, long term scheduler will be invoked again.
  - Invocation interval of seconds, minutes or longer.
  - In some systems, there may not be a long-term scheduler: Unix for example.

Schedulers

- Short-term scheduler:
  - Selects from among the processes in ready queue.
  - Invoked much more frequently.
    - At least once every 100ms.
  - Therefore, short-term scheduler should be fast in picking up the next process to execute.
- Midterm-schedulers:
  - Used in some systems.
  - Can be used to store executing programs to the disk for some time and then reload into memory: called swapping.
    - This can increase the process mix that are executing.
    - Can be used when processes' memory requirements are more than the physical memory size.

Medium-term scheduling model



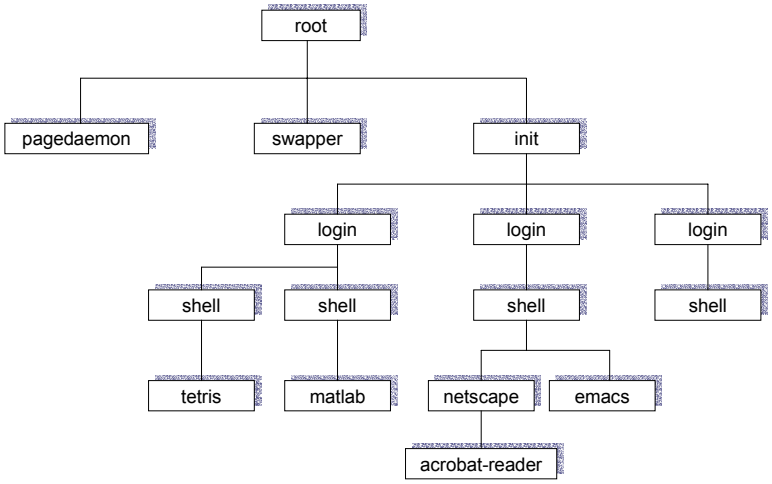
Operations on Processes

- Some operations on processes are:
  - Creating a process
  - Terminating a process
  - Waiting for a child to terminate
- OS must provide facilities (mechanisms) for these operations.

Operations on Processes: Process Creation

- A process may create several other processes
  - The created processes are called children
  - The creating process is called the parent
- In this way, a *tree* of processes may be created.
- Each process has an ID (pid)
  - The parent knows the its children's IDs.

A tree of processes in UNIX system



Operations on Processes: Process Creation

- When a child created:
  - In Unix, The memory image of a parent process is duplicated and the child own the duplicate.
    - This means:
      - All global variables of the child will have initial values set to the values of global variables of the parent at child creation time.
        - When child and parent continue execution, these variables may be set to different values.
      - This child will inherit all the open file descriptors of the parent.
        - This means, descriptors of the parent is copied into the child.
        - This means, parent and child can access the same files or devices.

Operations on Processes: Process Creation

- When a process creates a new child process, there are two possibilities exists in terms of execution of the parent:
  - The parent continues execute concurrently with its children
  - The parent waits until some or all of its children have terminated.
- There are also two possibilities in terms of the address space of the child:
  - The child process is duplicate of the parent process
    - UNIX, Windows
  - The child process has a program loaded into it (Windows)
    - Windows

Example: Unix process creation

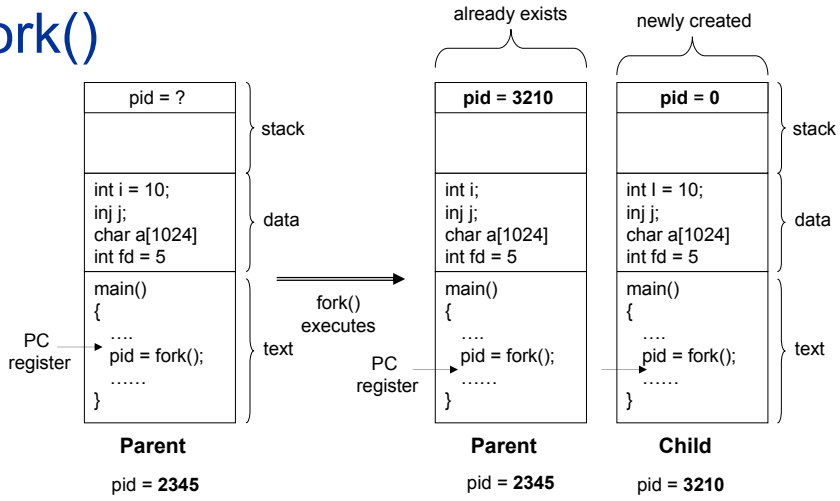
- In Unix, the only way to create a new process is calling the fork() system call.
- In Unix, each process has a process identifier (which is an integer value).
- fork() system call creates a exact duplicate of the calling (parent) process.
  - The child then can load a new program if it wants to execute some different program than what the parent process executes.
    - Can do this by use of execlp system call
    - execlp replaces the memory image of a process with some new program.

Example: Unix process creation

```
#include <stdio.h>
int i, j,  char a[1024];  int fd;
main(int argc, char *argv[])
{
        int pid;

        fd = fopen("myfile.txt", R_ODLY);
        pid = fork();        /* create a new process */
        if (pid < 0) {       /* system call is not successful */
                fprintf(stderr, "fork() failed\n");
                exit(-1);  /* termination with some error */
        }
        else if (pid == 0) {
                /* this is the child process */
                execlp ("/bin/ls", "ls", NULL);
        }
        else {
                /* this is the parent process */
                wait(NULL);          /* wait for the child to terminate */
                printf("child completed\n");
                exit(0);   /* normal termination */
        }
}
```

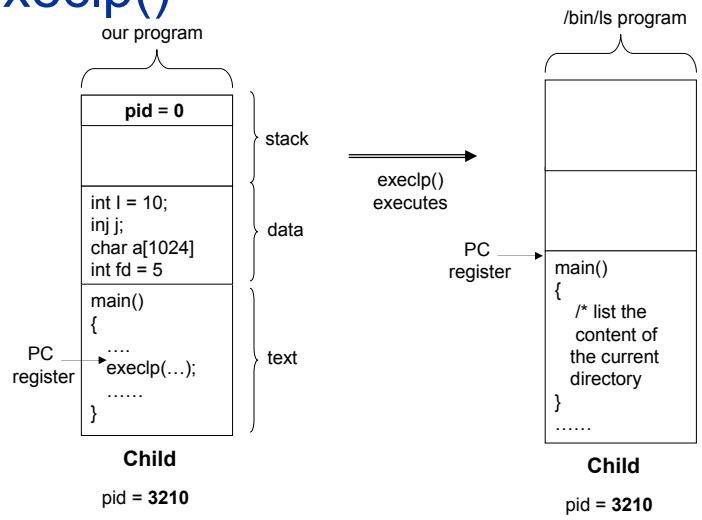
fork()



Before calling fork()

After calling fork()

execlp()



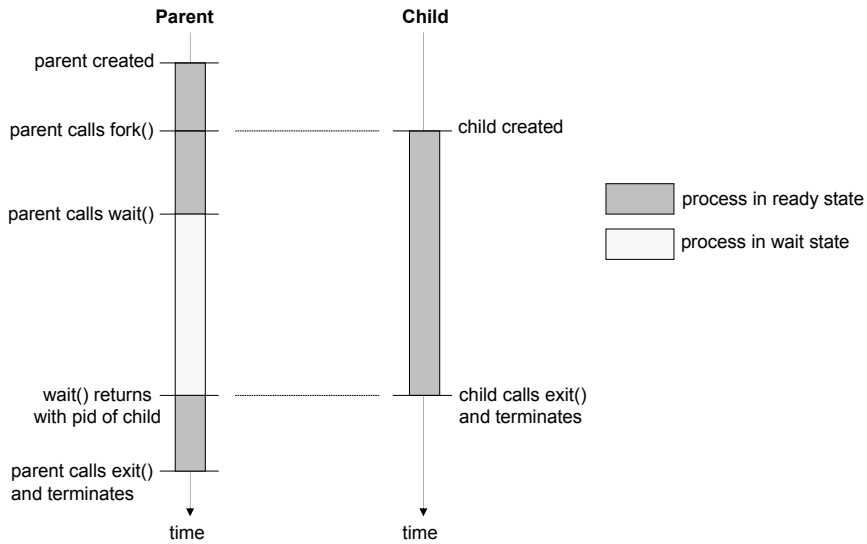
Before calling execlp()

After calling execlp()

Process termination

- When a process terminates, all the resources of the process are deallocated by the OS.
  - Memory
  - Open files.
  - I/O buffers
  - …
- A process may indicate to OS that it wants to terminate by calling the exit() system call.
  - At that point, the process may return some information (such as the exit code) to its parent, if the parent is waiting for the process to terminate.
- A parent may wait for a child to terminate by calling the wait() ot waitpid() system call.
- If a parent does not wait for a child to terminate, the parent can simply terminate after creating the child and optionally doing some more job.
  - In this case, the parent of the child will be a default process (init process in Unix)

Parent waits for child to terminate



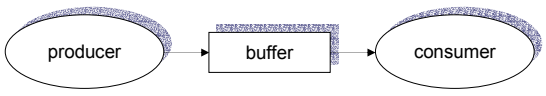
Cooperating Processes

- Processes in a system can be:
  - Independent: no sharing of data and flow of information between processes
  - Cooperating: processes work together to achieve a job. They may exchange data.
- Need for an environment that supports cooperation for:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience

Cooperating Processes

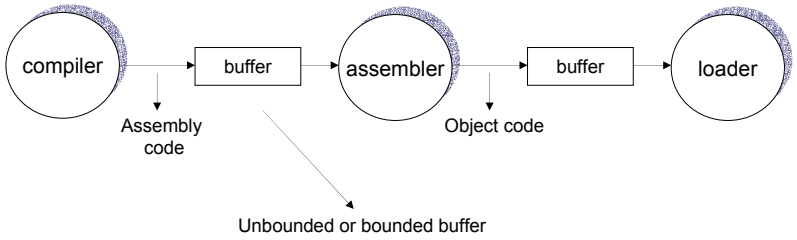
- Concurrent execution requires
  - Communication facilities
  - Synchronization of actions of more than one processes.

Producer-consumer problem: paradigm for programming concurrent and cooperating processes.



Cooperating Processes

- Producer consumer problem examples:
  - Print program – printer driver
  - Compiler – assembler - loader



Cooperating Processes

- Buffer could be
  - A shared memory between consumer(s) and producer(s).
- Could be provided by OS through the use of interprocess-communication (IPC) facilities.
- We will later see:
  - Interprocess communication
  - Synchronization