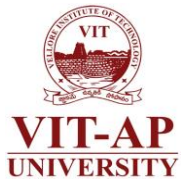
CSE2008: Operating Systems

L7 & L8: Process & Programs



Dr. Subrata Tikadar

SCOPE, VIT-AP University

Recap

- Introductory Concepts
  - What Operating Systems Do?
  - Computer System Organization
  - Computer-System Architecture
  - Operating-System Operations
  - Resource Management
  - Security and Protection
  - Virtualization
  - Distributed Systems
  - Kernel Data Structures
  - Computing Environments
  - Free and Open-Source Operating Systems
  - Overview of System Calls

Outline

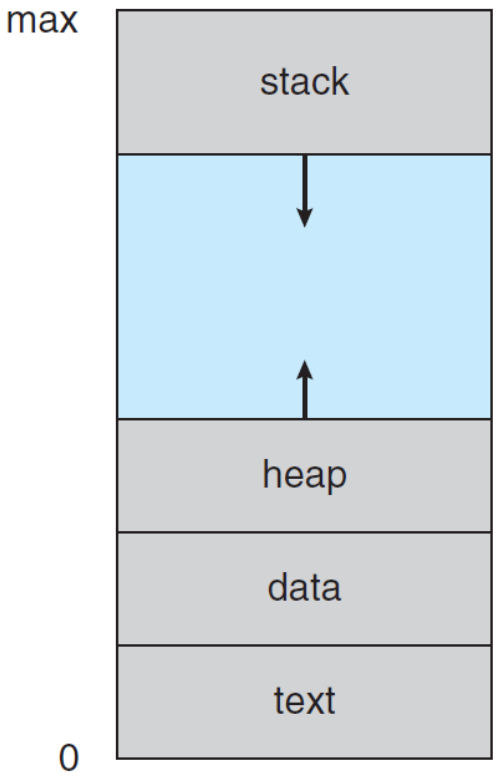
- Process Concept
- Process Scheduling
- Operations on Processes

Process Concept

- The Process
- Process State
- Process Control Block

Process Concept

- The Process



← temporary data storage when invoking functions
(such As function parameters, return addresses, and local variables)

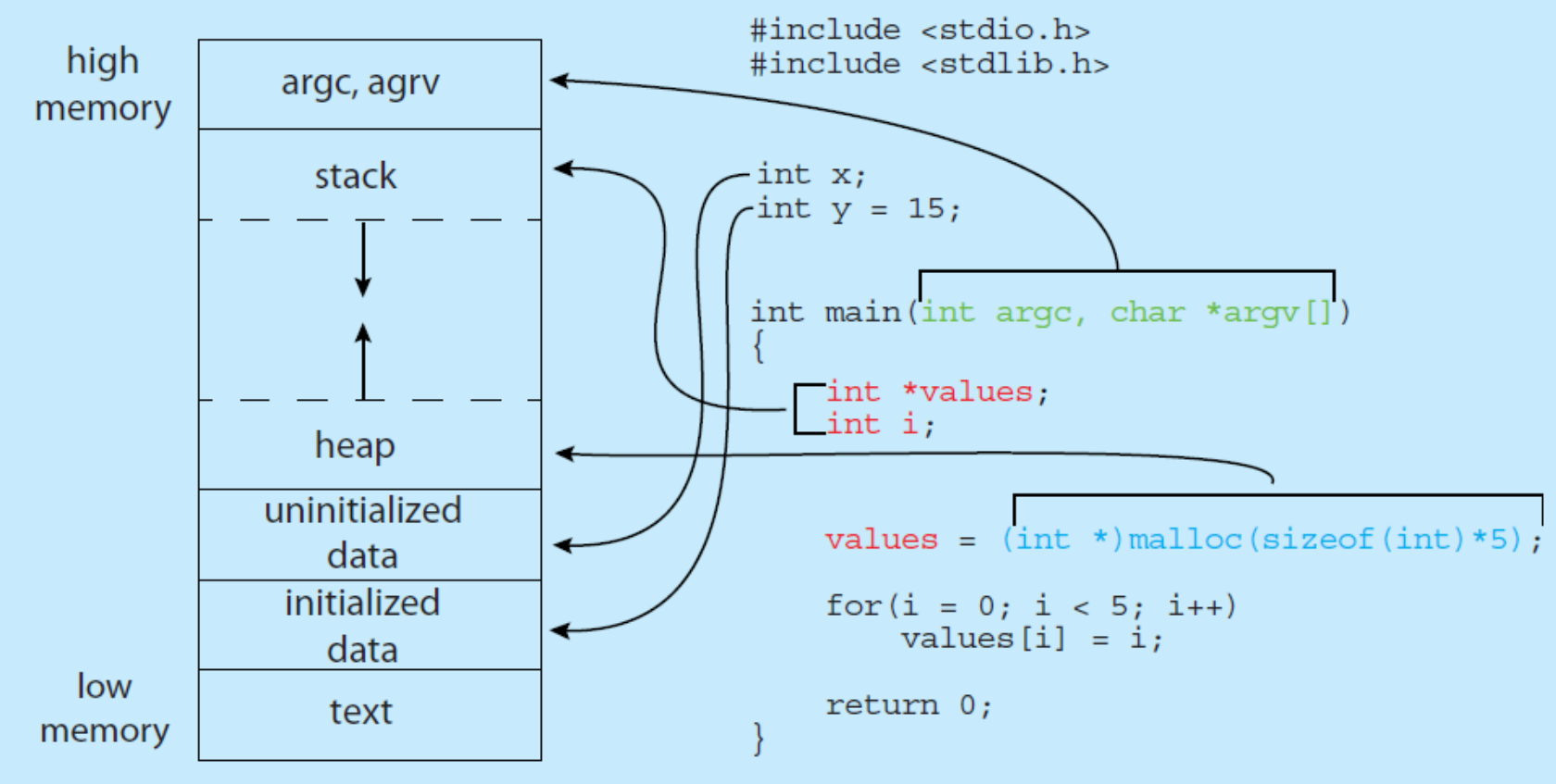
← memory that is dynamically allocated during program run time

← global variables

← the executable code

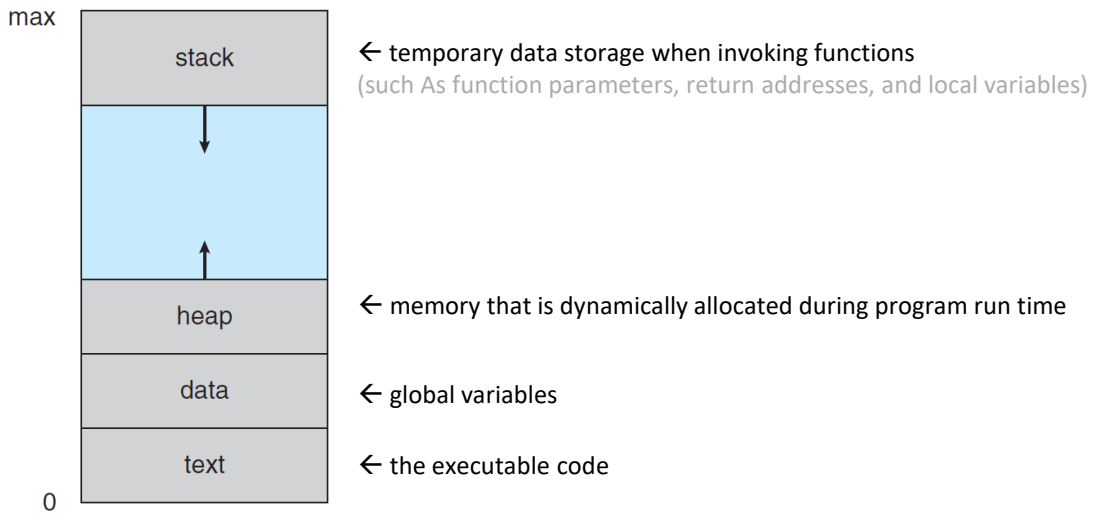
Process Concept

- The Process



Process Concept

- The Process



Note:

1. A program is a passive entity, such as a file containing a list of instructions stored on disk; whereas, a process is an active entity, with a program counter specifying the next instruction to execute and a set of associated resources.

2. A program becomes a process when an executable file is loaded into memory.

Double-clicking an icon representing the executable file

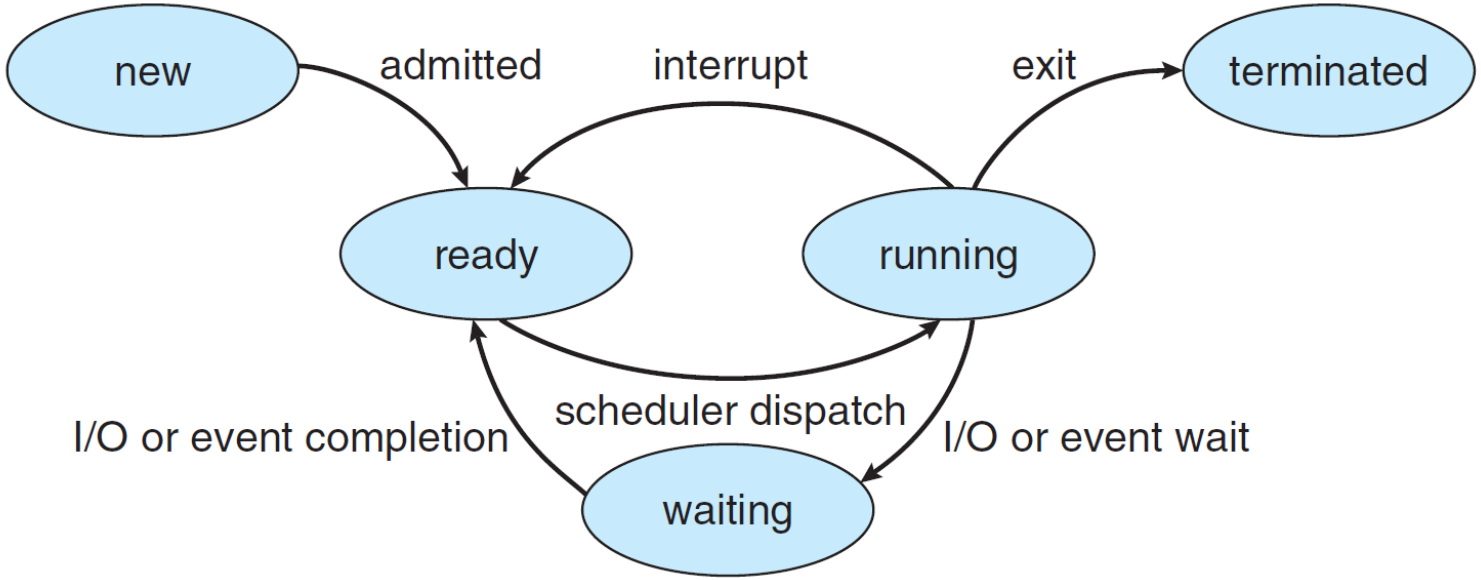
Entering the name of the executable file on the command line e.g., prog.exe / a.out

Process Concept

- Process State
  - **New:** The process is being created.
  - **Running:** Instructions are being executed.
  - **Waiting:** The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
  - **Ready:** The process is waiting to be assigned to a processor.
  - **Terminated:** The process has finished execution.

Process Concept

- Process State



New: The process is being created.

Running: Instructions are being executed.

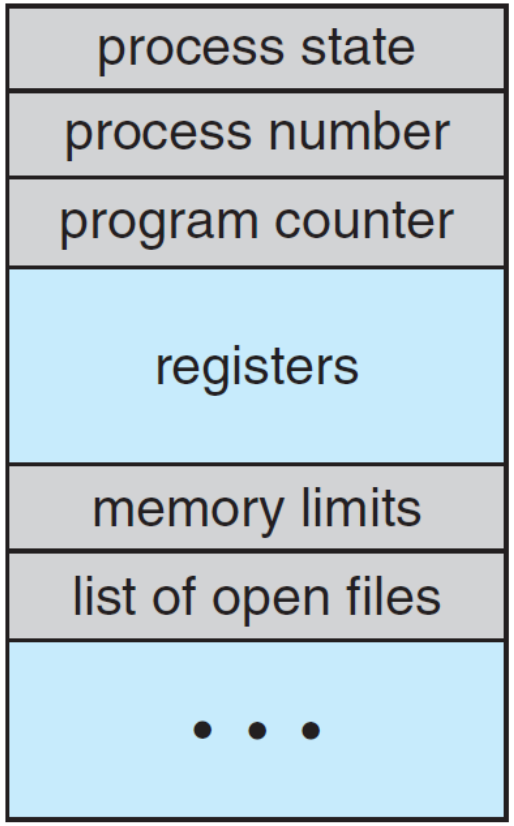
Waiting: The process is waiting for some event to occur (such as an I/O completion or reception of a signal).

Ready: The process is waiting to be assigned to a processor.

Terminated: The process has finished execution.

Process Concept

- Process Control Block



← new / ready / running / waiting / halted …

← Process ID

← indicates the address of the next instruction to be executed for the process

← accumulators, index registers, stack pointers, and general-purpose registers, etc.

← value of the base and limit registers and the page tables / segment tables

← list of I/O devices allocated to the process, a list of open files, etc.

Process Concept

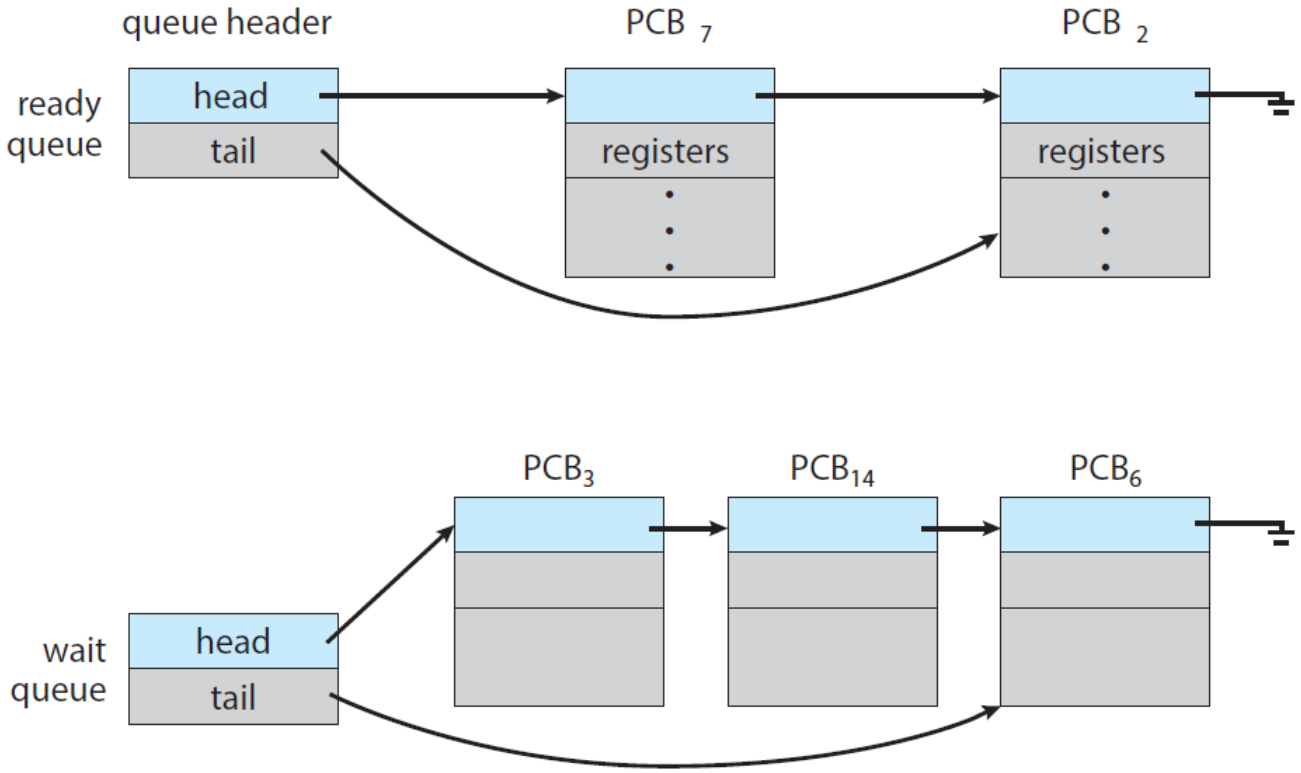
- Threads
  - Single thread of control
    - allows the process to perform only one task at a time
  - Multiple threads of execution
    - perform more than one task at a time

Process Scheduling

- The objective of multiprogramming is to have some process running at all times so as to maximize CPU utilization
- The objective of time sharing is to switch a CPU core among processes so frequently that users can interact with each program while it is running
  - To meet these objectives, the process scheduler selects an available process (possibly from a set of several available processes) for program execution on a core
  - Each CPU core can run one process at a time
    - For a system with a single CPU core, there will never be more than one process running at a time, whereas a multicore system can run multiple processes at one time. If there are more processes than cores, excess processes will have to wait until a core is free and can be rescheduled

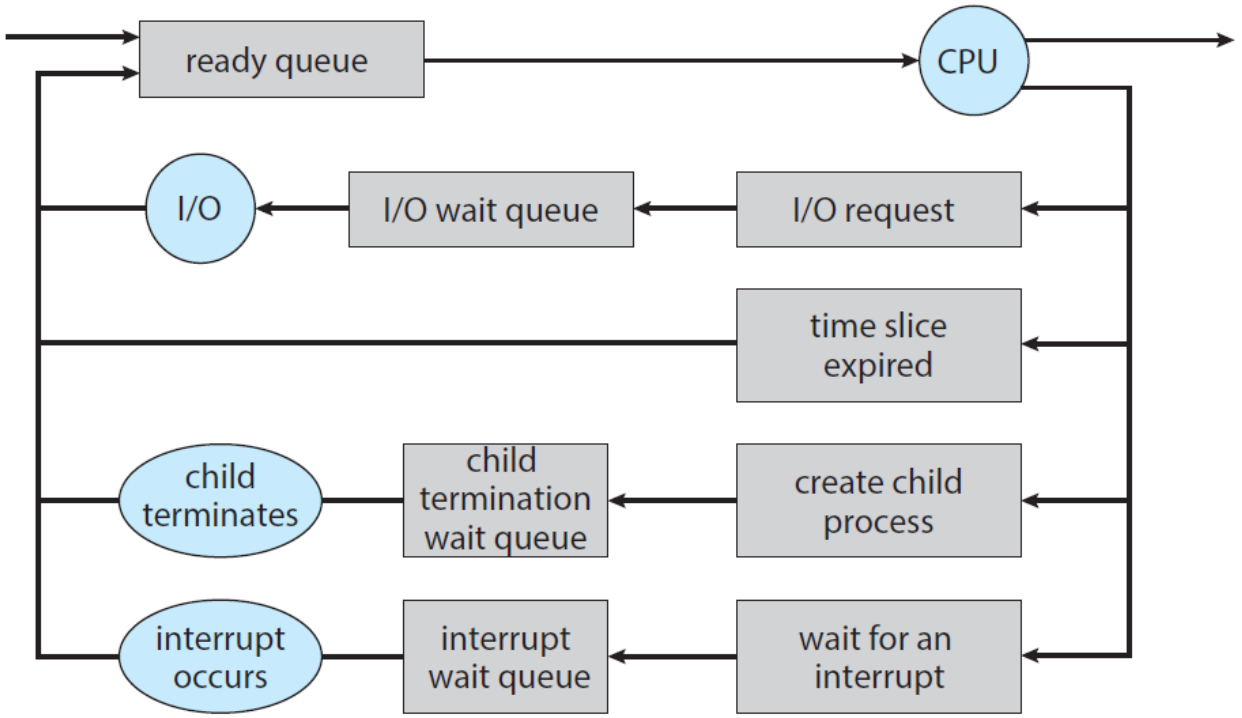
Process Scheduling

- Scheduling Queues



Process Scheduling

- Queueing-diagram representation of process scheduling

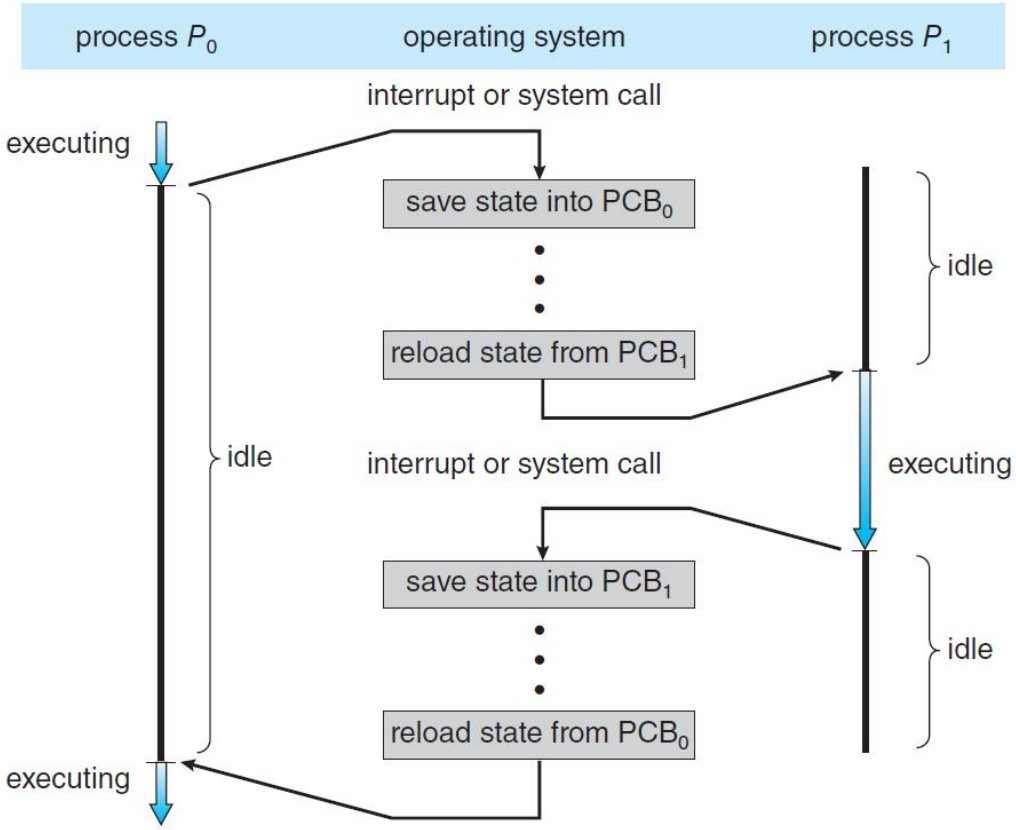


Process Scheduling

- CPU Scheduling
  - CPU scheduler selects from among the processes that are in the ready queue and allocate a CPU core to one of them
  - Some OSs have an intermediate form of scheduling, known as swapping
    - Sometimes it can be advantageous to remove a process from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming

Process Scheduling

- Context Switch

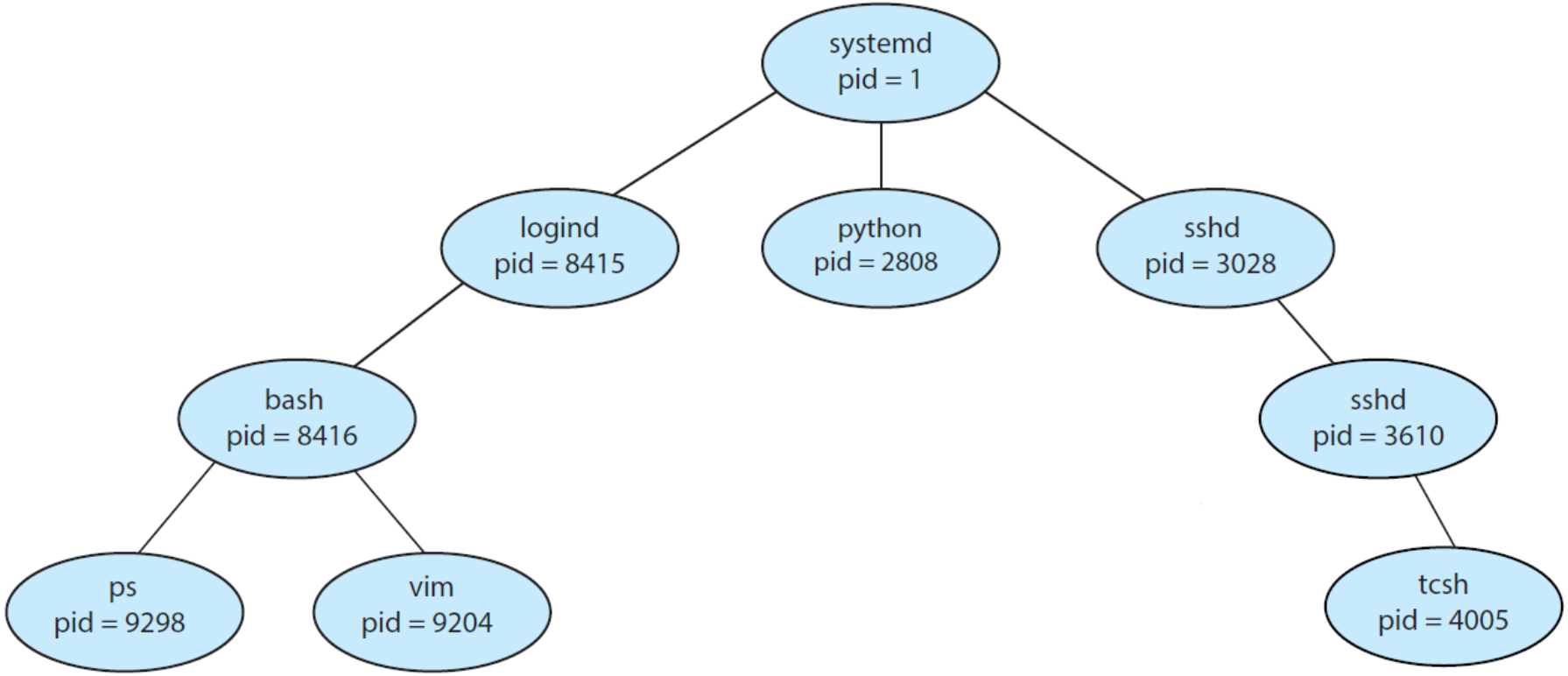


Operations on Processes

- Process Creation
- Process Termination

Operations on Processes

- Process Creation



Operations on Processes

- Process Creation
  - Two possibilities for execution of a new process
    1. The parent continues to execute concurrently with its children.
    2. The parent waits until some or all of its children have terminated.
  - Two address-space possibilities for the new process
    1. The child process is a duplicate of the parent process (it has the same program and data as the parent).
    2. The child process has a new program loaded into it

Operations on Processes

- Process Creation
  - Example [Creating a separate process using the UNIX fork() system call]

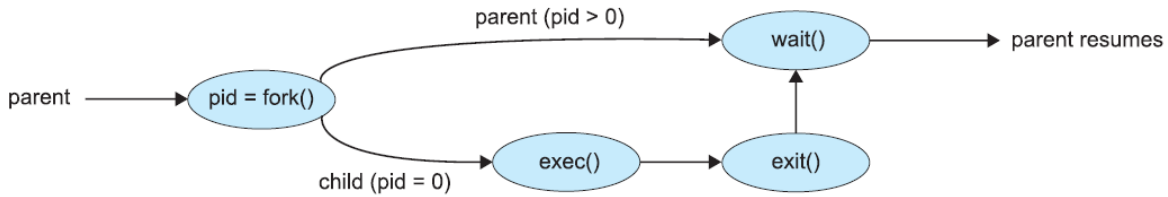
```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
    }

    return 0;
}
```



Operations on Processes

- Process Creation
  - Example [Creating a separate process using the CreateProcess() system call (Windows based API)]

```
#include <stdio.h>
#include <windows.h>

int main(VOID)
{
STARTUPINFO si;
PROCESS_INFORMATION pi;

   /* allocate memory */
   ZeroMemory(&si, sizeof(si));
   si.cb = sizeof(si);
   ZeroMemory(&pi, sizeof(pi));

   /* create child process */
   if (!CreateProcess(NULL, /* use command line */
    "C:\\WINDOWS\\system32\\mspaint.exe", /* command */
    NULL, /* don't inherit process handle */
    NULL, /* don't inherit thread handle */
    FALSE, /* disable handle inheritance */
    0, /* no creation flags */
    NULL, /* use parent's environment block */
    NULL, /* use parent's existing directory */
    &si,
    &pi))
   {
     fprintf(stderr, "Create Process Failed");
     return -1;
   }
   /* parent will wait for the child to complete */
   WaitForSingleObject(pi.hProcess, INFINITE);
   printf("Child Complete");

   /* close handles */
   CloseHandle(pi.hProcess);
   CloseHandle(pi.hThread);
}
```

Operation

- Process Creati
  - Example [Cre n call (Windows based API)]

```
#include <stdio.h>
#include <windows.h>

int main(VOID)
{
STARTUPINFO si;
PROCESS_INFORMATION pi;

    /* allocate memory */
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    /* create child process */
    if (!CreateProcess(NULL, /* use command line */
     "C:\\WINDOWS\\system32\\mspaint.exe", /* command */
     NULL, /* don't inherit process handle */
     NULL, /* don't inherit thread handle */
     FALSE, /* disable handle inheritance */
     0, /* no creation flags */
     NULL, /* use parent's environment block */
     NULL, /* use parent's existing directory */
     &si,
     &pi))
    {
       fprintf(stderr, "Create Process Failed");
       return -1;
    }
    /* parent will wait for the child to complete */
    WaitForSingleObject(pi.hProcess, INFINITE);
    printf("Child Complete");

    /* close handles */
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

Operations on Processes

- Process Termination
  - Reasons for process termination
    - The child has exceeded its usage of some of the resources that it has been allocated (to determine whether this has occurred, the parent must have a mechanism to inspect the state of its children)
    - The task assigned to the child is no longer required
    - The parent is exiting, and the operating system does not allow a child to continue if its parent terminates

---

```
/* exit with status 1 */
exit(1);
```

```
pid_t pid;
int status;
pid = wait(&status);
```

Android Process Hierarchy

- **Foreground process**—The current process visible on the screen, representing the application the user is currently interacting with
- **Visible process**—A process that is not directly visible on the foreground but that is performing an activity that the foreground process is referring to (that is, a process performing an activity whose *status* is displayed on the foreground process)
- **Service process**—A process that is similar to a background process but is performing an activity that is apparent to the user (such as streaming music)
- **Background process**—A process that may be performing an activity but is not apparent to the user.
- **Empty process**—A process that holds no active components associated with any application

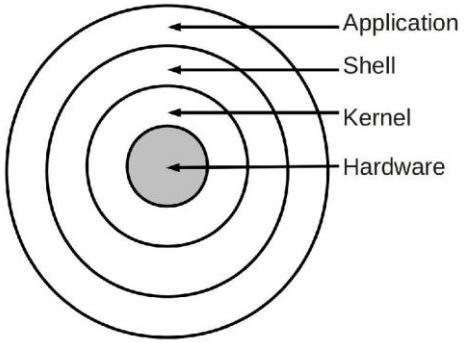
Reference

- Abraham Silberschatz , Peter B. Galvin, Greg Gagne, “Operating System Concepts”, Addison Wesley, 10th edition, 2018
  - Chapter 3: Section 3.1 – 3.3

Next

- IPC

Appendix



- Operating system shells
  - Your interface to the operating system is called a shell.
  - The shell is the outermost layer of the operating system. Shells incorporate a programming language to control processes and files, as well as to start and control other programs. The shell manages the interaction between you and the operating system by prompting you for input, interpreting that input for the operating system, and then handling any resulting output from the operating system.
  - Shells provide a way for you to communicate with the operating system. This communication is carried out either interactively (input from the keyboard is acted upon immediately) or as a shell script. A shell script is a sequence of shell and operating system commands that is stored in a file.
  - When you log in to the system, the system locates the name of a shell program to execute. After it is executed, the shell displays a command prompt. This prompt is usually a $ (dollar sign). When you type a command at the prompt and press the Enter key, the shell evaluates the command and attempts to carry it out. Depending on your command instructions, the shell writes the command output to the screen or redirects the output. It then returns the command prompt and waits for you to type another command.
  - Types:
    - Korn shell: The Korn shell (ksh command) is backwardly compatible with the Bourne shell (bsh command) and contains most of the Bourne shell features as well as several of the best features of the C shell.
    - Bourne shell: The Bourne shell is an interactive command interpreter and command programming language.
    - C shell: The C shell is an interactive command interpreter and a command programming language. It uses syntax that is similar to the C programming language.

VIT-AP UNIVERSITY

Thank You