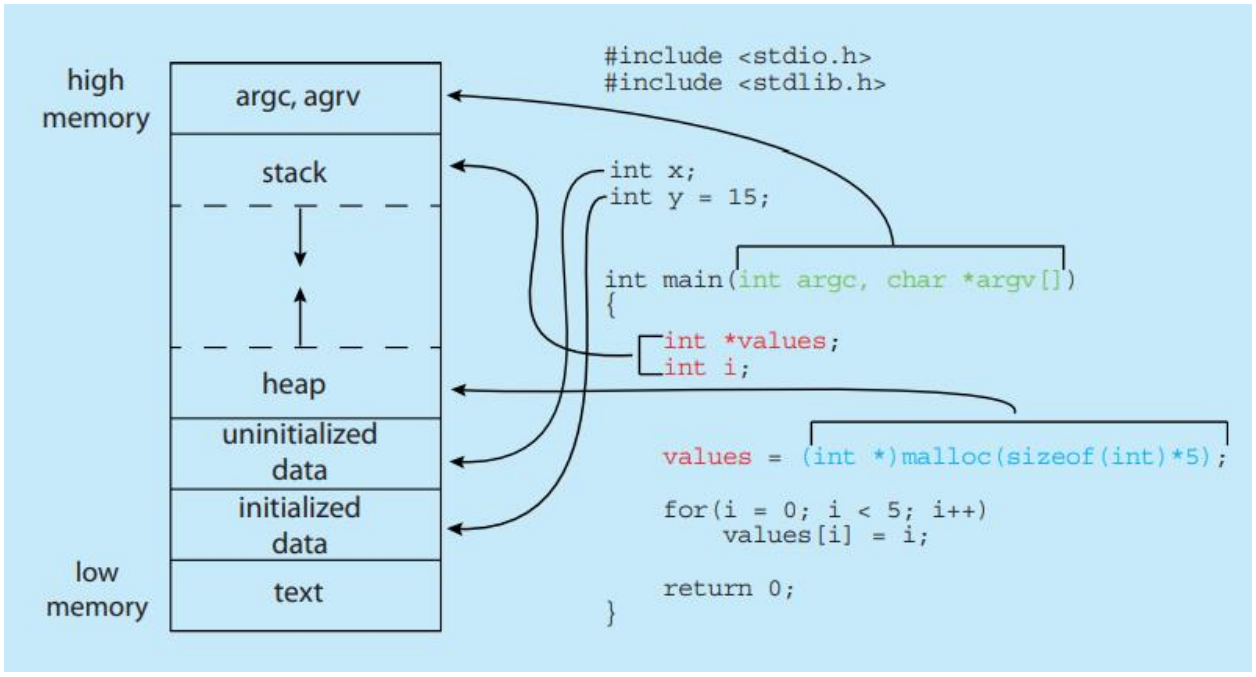
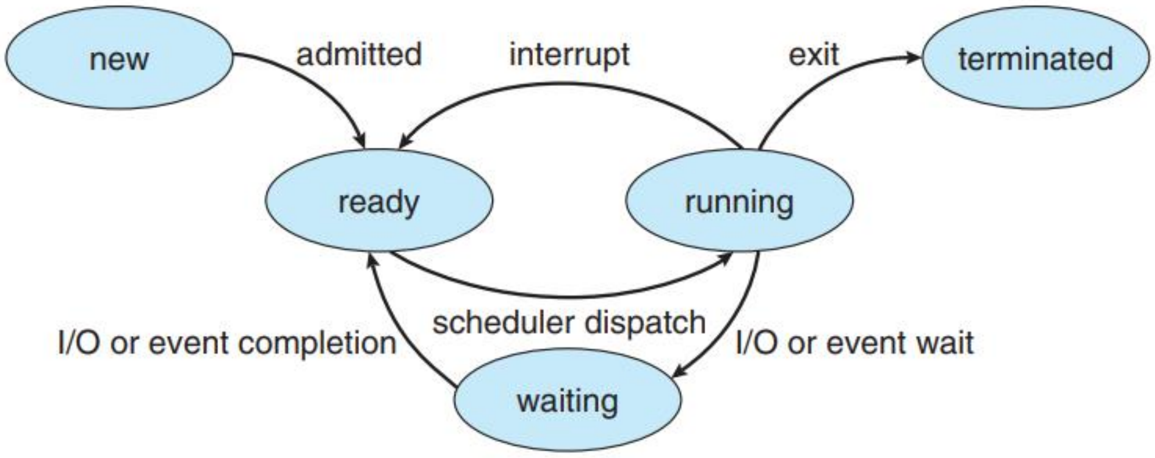
3.1

- Programs are passive entities stored in the secondary memory as files of sequential instructions and processes are active entities utilizing space in the main memory
- The area allocated to a program in the main memory is divided into text, data, heap, and stack sections
- Text refers to actual text (code) of the program
- Data refers to the global variables
- Heap is used to store dynamically allocated variables
- Stack is used to store the activation record (function parameters, local variables, and return address) of function calls
- Text and Data sections are static and Heap and Stack sections are dynamic (can grow)



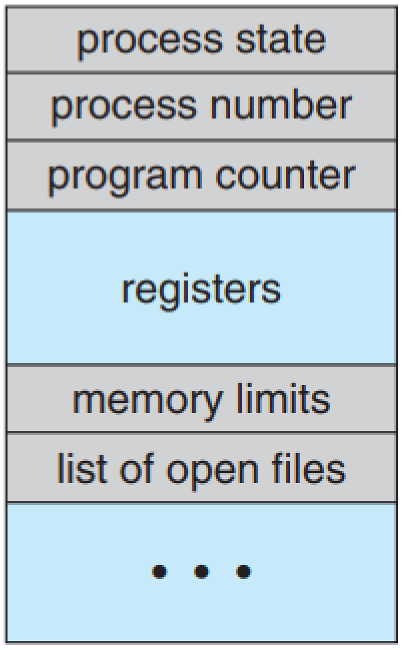
```
#include <stdio.h>
#include <stdlib.h>

int x;
int y = 15;

int main(int argc, char *argv[])
{
    int *values;
    int i;

    values = (int *)malloc(sizeof(int)*5);

    for(i = 0; i < 5; i++)
        values[i] = i;

    return 0;
}
```

- The above diagram shows that command line arguments also have separate space allocated for them given the user utilizes them
- As a process executes it goes through many states
- New State: Process as it is being created
- Ready State: When the process is waiting to be assigned to a processor for execution
- Running State: The state of execution
- Waiting State: When a process is waiting for an event (I/O completion)
- Terminated State: the state of the process as it ends



- The Process Control Block (PCB) is a struct/object that is used in process scheduling
- It contains the process ID, program counter, registers, the process state, and other information
- The process ID is the unique number assigned to each process to distinguish them from other processes

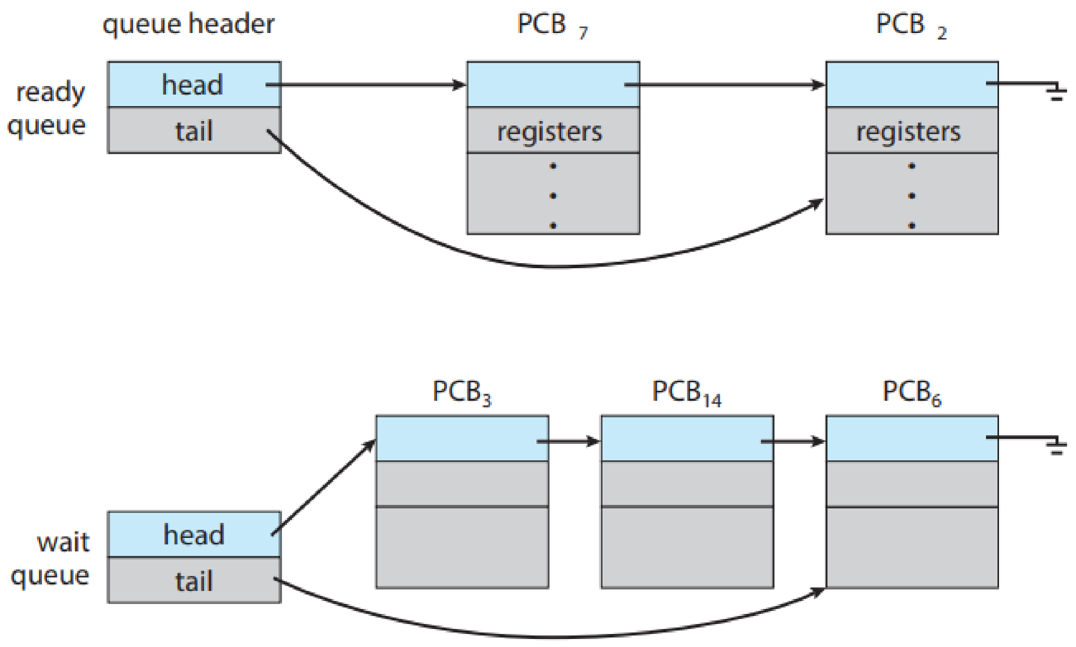
- The program counter contains the address of the next instruction to be executed
- The registers store all the important data needed in context switching etc.
- The process state refers to a particular stage in the execution for a process (ready, new, terminated, waiting, running, etc.)

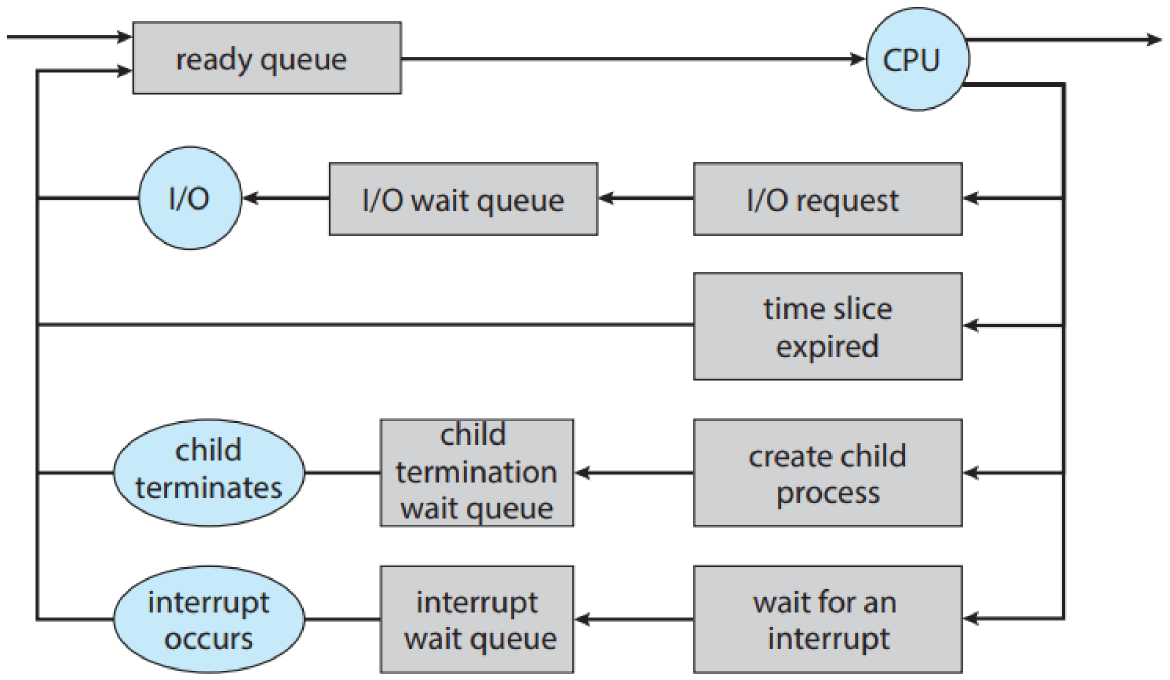


- Threads refer to multiple instruction sequences that are executed in parallel
- A program with multiple threads has multiple program counters
- An example would be a word processer that takes input and performs syntax checks in parallel

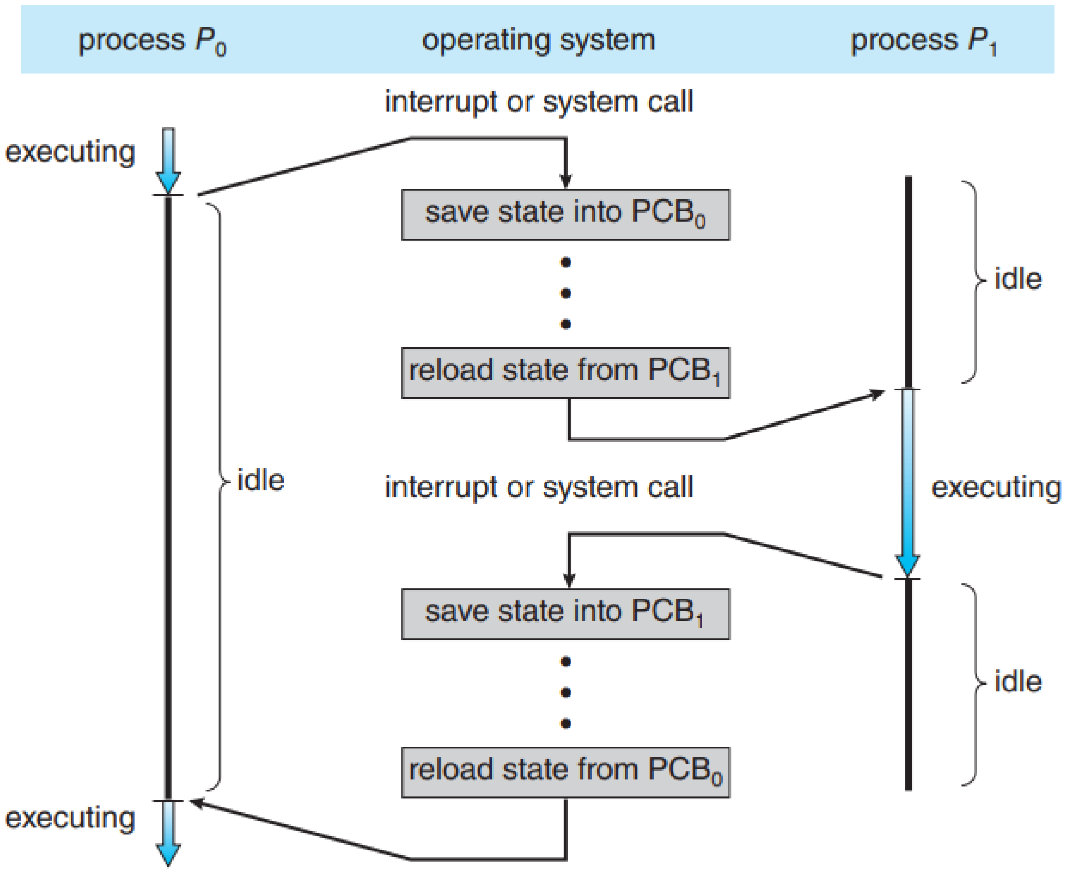
3.2

- The number of processes currently in memory is known as the degree of multi-programming
- An I/O bound process is a process that requests for I/O devices frequently whereas a CPU bound process utilizes most of its time in computations
- A CPU can only execute a single process at a time and therefore if the number of processes exceed the number of cores than they are managed through multiple queues implemented through linked lists





- Processes that are waiting for an I/O request completion are placed in the I/O wait queue
- Processes that create a child process and wait for its execution to complete are placed in the child termination queue
- Processes that are waiting for interrupts (from some other process or I/O etc.) are placed in the interrupt wait queue
- A CPU scheduler manages the current process being run in the CPU by swapping it out or placing it in different queues to load other process into the core to maintain efficiency
- A context switch is when a process’s context (stored in the PCB) is stored elsewhere and a new process is loaded into the core
- The saving of a context is called a state save and the restoring of a context is called a state restore (you are essentially saving and restoring the state of the CPU each time)

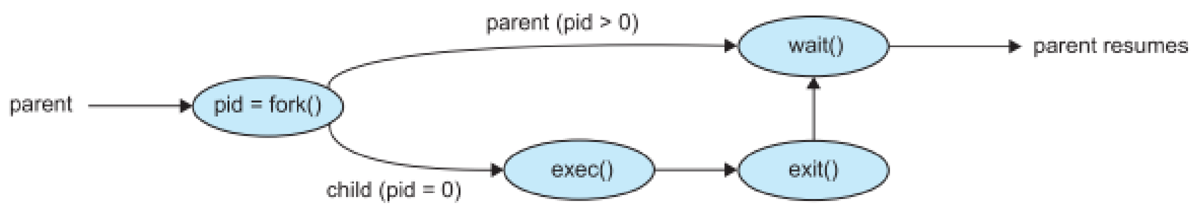

process P_0
operating system
process P_1
interrupt or system call
executing
save state into PCB_0
idle
reload state from PCB_1
idle
interrupt or system call
executing
save state into PCB_1
idle
reload state from PCB_0
executing

3.3

- The first process to run on the computer is either the Init process or the systemd process
- One or the other of the two initial processes becomes the root process for a tree of processes
- The child process can get separate memory allocated for itself through the OS or it can occupy a subset of the resources allocated for its parent processes
- The parent and the child may execute concurrently or the parent may be placed in the child termination wait queue to wait for the child’s termination
- The fork system call creates a child process which is a copy of its parent process and starts executing the line immediately after the fork command
- Fork returns a zero identifier to the child process and a value greater than zero for the parent
- After the fork command, the execlp system call is used to replace the address space of one of the two processes by redirecting the process to a different program
- A parent may use the wait system call to halt its execution as the child process executes
- The child process asks to be deleted by calling the exit system call which returns an integer to the parent which deletes the child as the operating system deallocates its resources
- The exit status and process identifier of the child process are transferred to the parent through the wait instruction after it terminates so that it could be removed from the process table

- The child process having been terminated but not removed from the process table is called a zombie process
- The child process, whose parent terminates before it is left as an orphan process

```
#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;
}
```


parent (pid > 0)
wait()
parent resumes
parent
pid = fork()
exec()
exit()
child (pid = 0)