COMPUTER ORGANIZATION AND SOFTWARE SYSTEMS

Session 9

Prof. C R Sarma

BITS Pilani

Pilani Campus

Today's Session

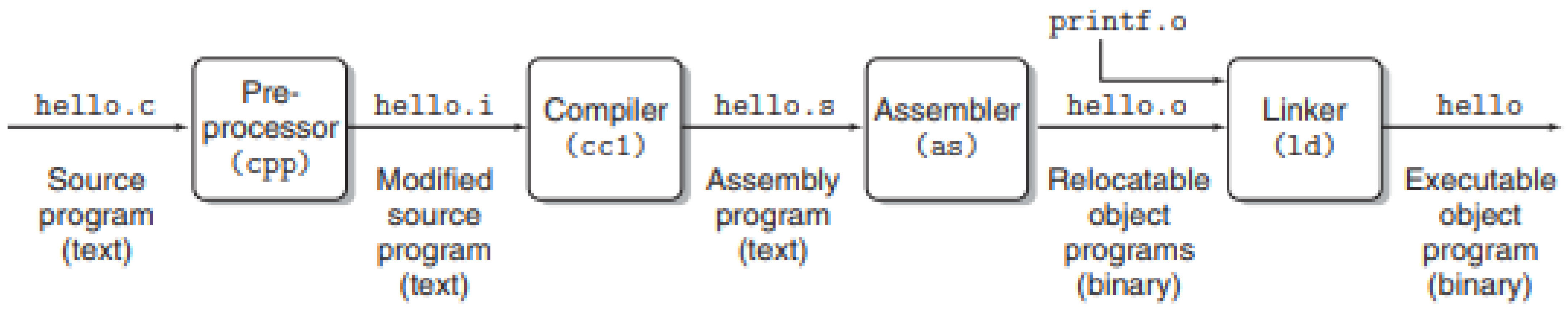
Contact Hour	List of Topic Title	Text/Ref Book/external resource
CS9	**Process Management** (T2: 3.1-3.3 and 4.1-4.3) • Concept of Process • Process State Diagram • Operations on Processes : Process creation and termination examples • Process vs. Threads • Multithreading Models	T2

Process Management

BITS Pilani
Pilani Campus

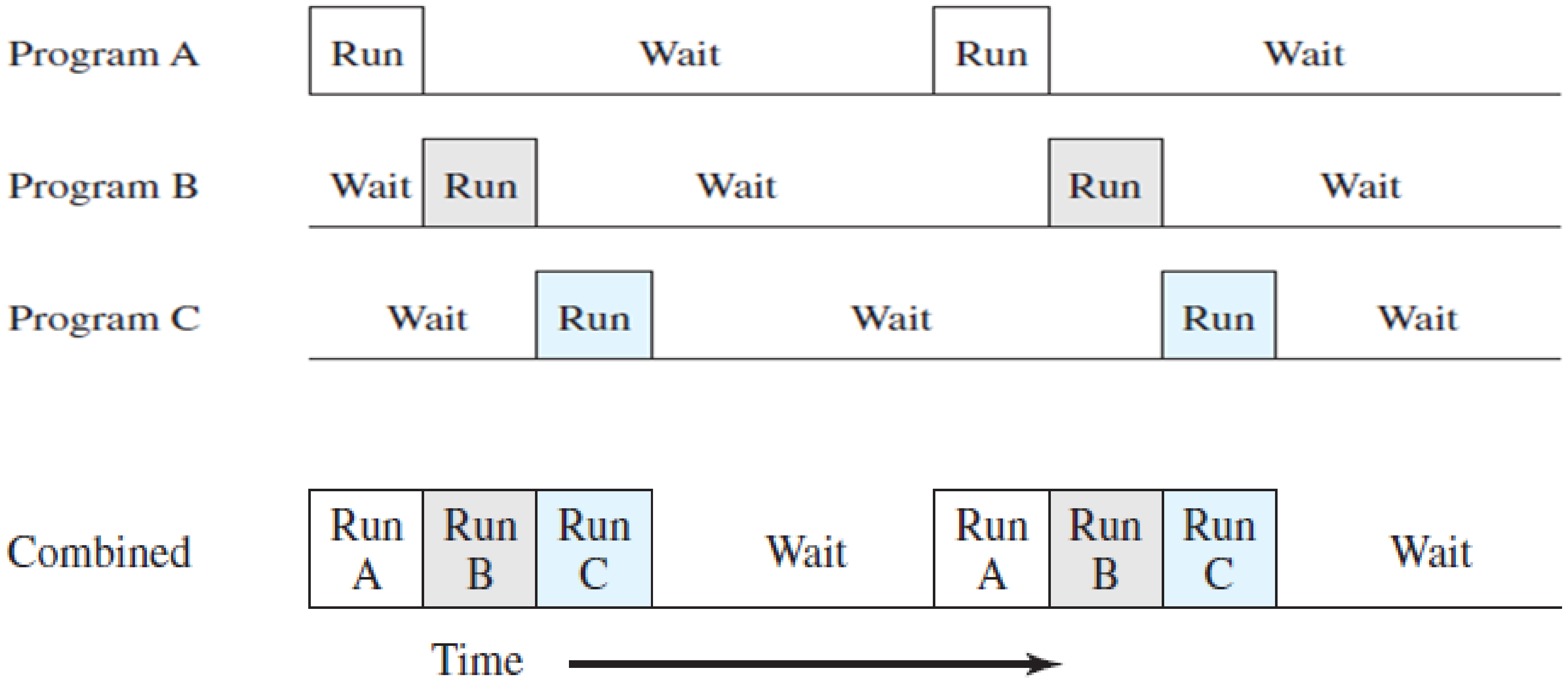
Introduction

- Program Execution



The compilation system.

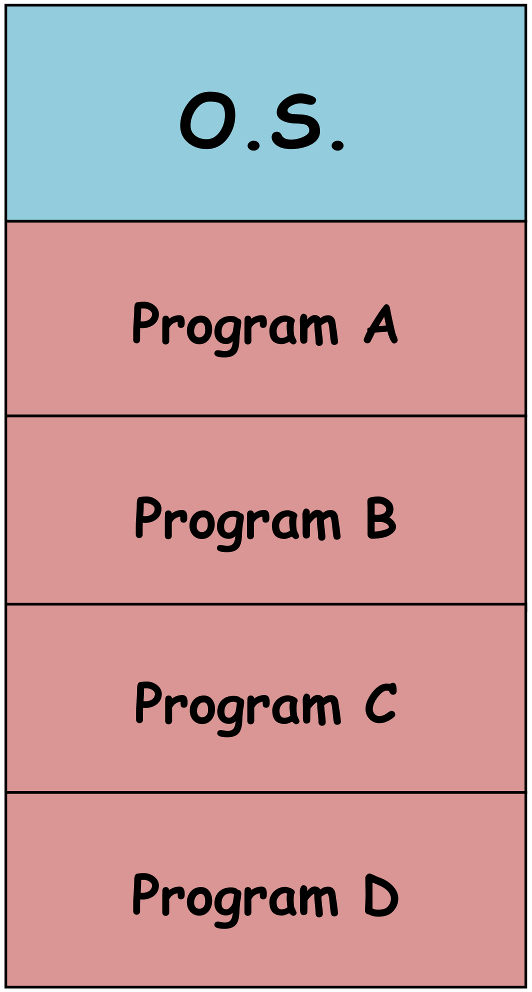
Multiprogramming vs Timesharing



Multiprogramming vs Timesharing

Memory layout of a

Multiprogramming system



Multiprogramming vs Timesharing

- Logical extension of multiprogramming systems
- Also known as multitasking
- Each user program is given a time slot to execute in round robin fashion
- Pseudo parallelism
- Time shared system can run several programs at the same time, so it is also called multiprogramming system. But multiprogramming is not a time shared system.
- Allows many users to share the computer resources simultaneously

Process

- ✓ A program in execution
- ✓ Needs resources : CPU, memory, files, I/O devices
- ✓ A program becomes process when executable file loaded into memory
- ✓ Process execution must progress in sequential fashion
- ✓ word processor, a Web browser and an e-mail package are different processes.
- ✓ Types: System processes and user processes

Process in Memory

- The text section: comprises the executable code.
- The data section: stores global and static variables, allocated and initialized prior to executing main.
- The heap: is used for dynamic memory allocation.
- The stack: is used for local variables. Space on the stack is reserved for local variables when they are declared and the space is freed up when the variables go out of scope. stack is also used for function return values.

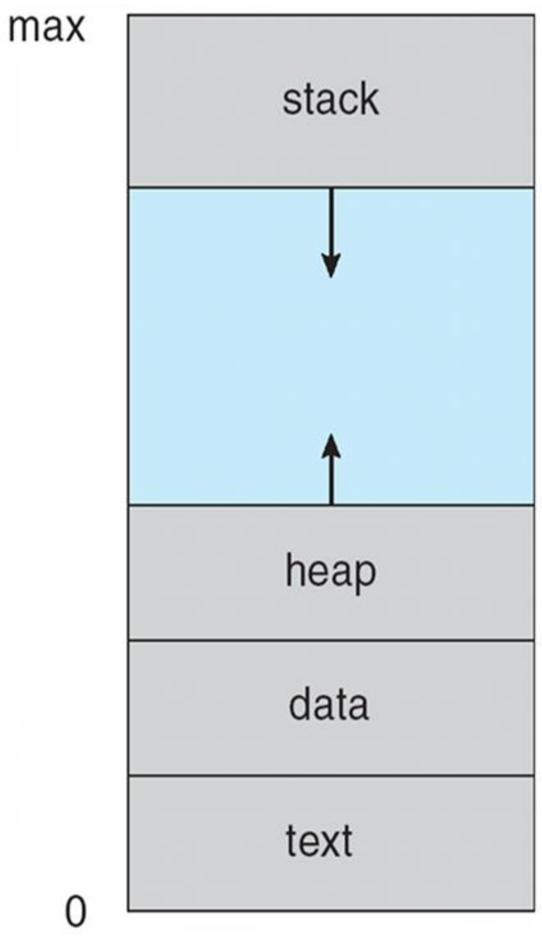
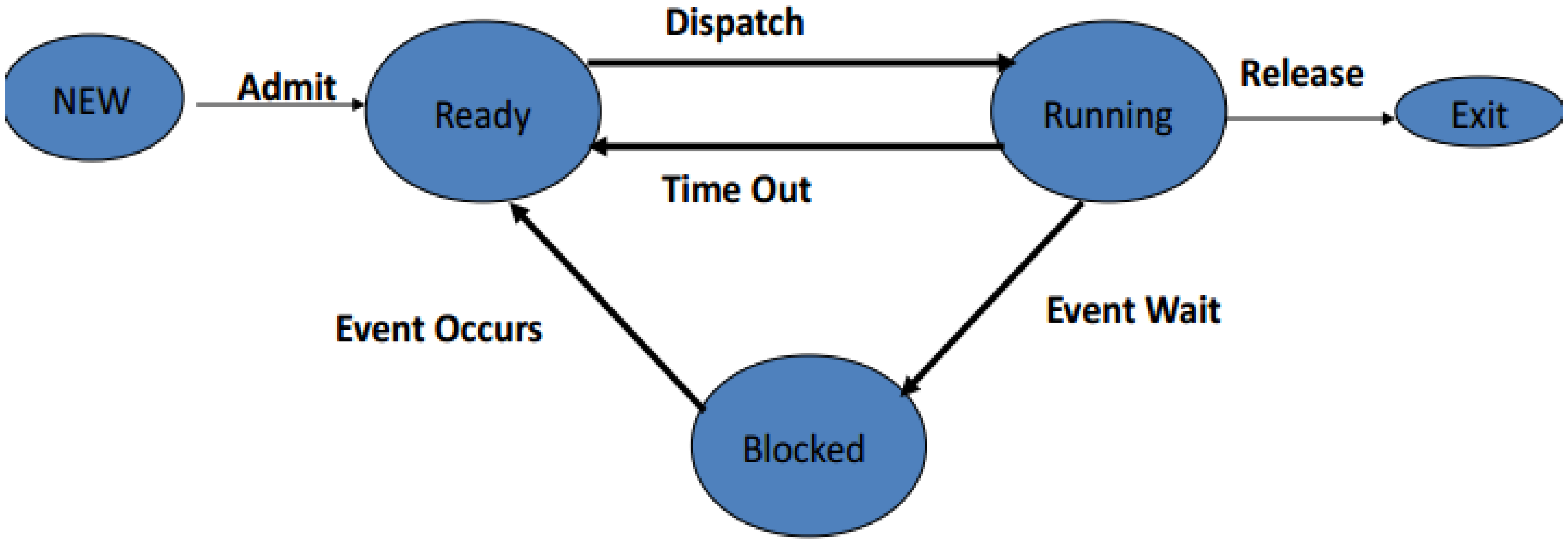


Diagram of Process State

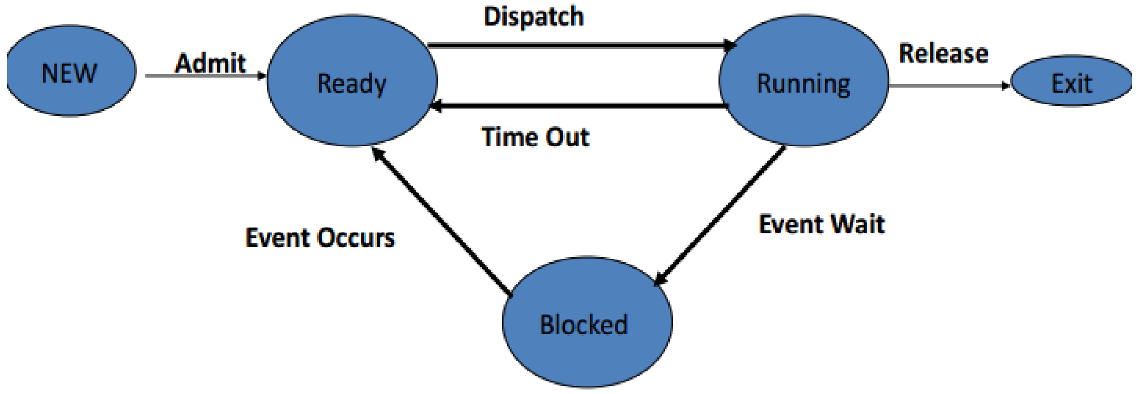


Process State

- Processes may be in one of 5 states
- **New** - The process is in the stage of being **created**.
- **Ready** – The process is **waiting to be assigned** to a processor
- **Running** – Instructions are being **executed** by the CPU.
- **Waiting/Blocked** - The process is **waiting for some resource** to become available. For example: keyboard input, disk access request etc.
- **Terminated** - The process has **finished** its task

Valid State Transitions

- **Null to new**
- New to ready
- Ready to running
- Running to exit
- Running to ready
- Running to blocked
- Blocked to ready
- **Ready to exit**
- **Blocked to exit**



Reasons for Process Creation

- New batch job
- Interactive logon
- Created by OS to provide a service
- Spawned by existing process

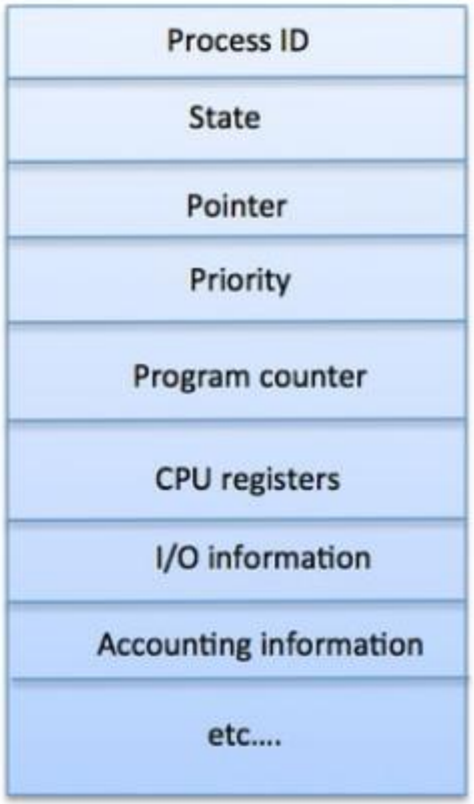
Reasons for Process Termination

- Normal Completion
- Time Limit Exceeded
- Memory Unavailable
- Bounds Violation
- Protection Error
- Arithmetic Error
- I/O Failure
- Invalid and Privileged Instruction
- Operator or OS Intervention
- Parent termination
- Parent request

Process Control Block

Each process has a PCB, which stores process-specific information like

- ✓ Process Number - Process Identifier
- ✓ Process State - Running, waiting, etc.
- ✓ Pointer – to parent, child (if any)
- ✓ Priority
- ✓ Program counter – location of instruction next to execute
- ✓ CPU registers – contents of all process-centric registers
- ✓ Accounting information – CPU used, clock time elapsed since start, time limits



Process Context Switch

- To move from one process to another on a context switch, we have to
  - **save the context** of the current process
  - **select the next** process to run
  - **restore the context** of this new process.
- **Context of a process**

  Program Counter, Stack Pointer, Registers, Code + Data + Stack (also called Address Space), Other state information maintained by the OS for the process (open files, scheduling info, I/O devices being used etc.)

Operations on Processes

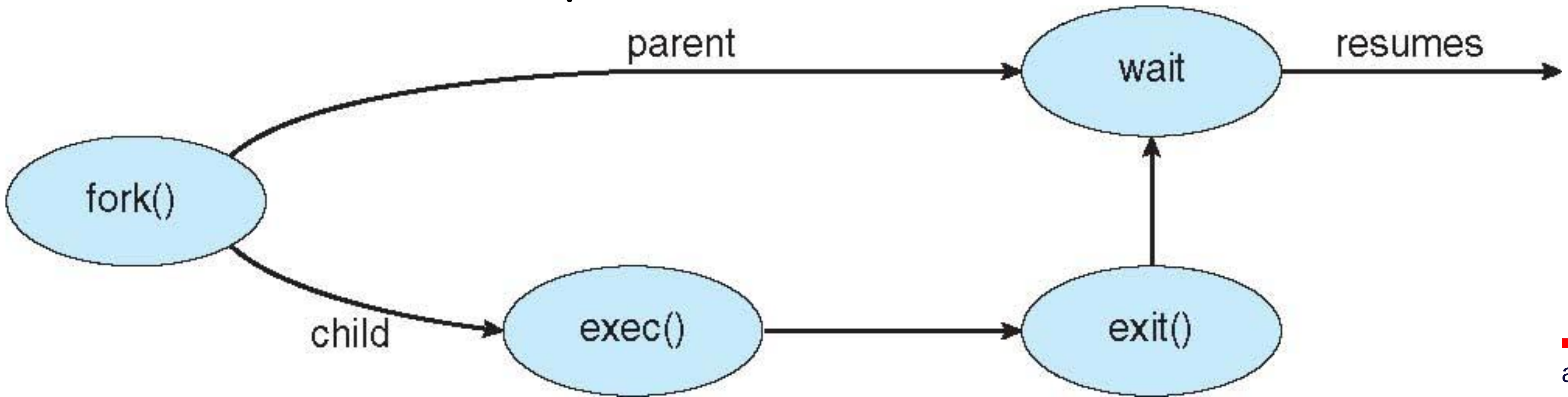
- Process Creation
- Process Termination
- Process Preemption and Blocking

Operations on Processes – Process Creation

- Created using system calls
  - Example : fork or spawn
  - Parent Vs Child
  - Each process stores PID ➔ Integer, and PPID
- **There are two options for the parent process after creating the child:**

  ✓ Wait for the child process to terminate

  ✓ Run concurrently with the child

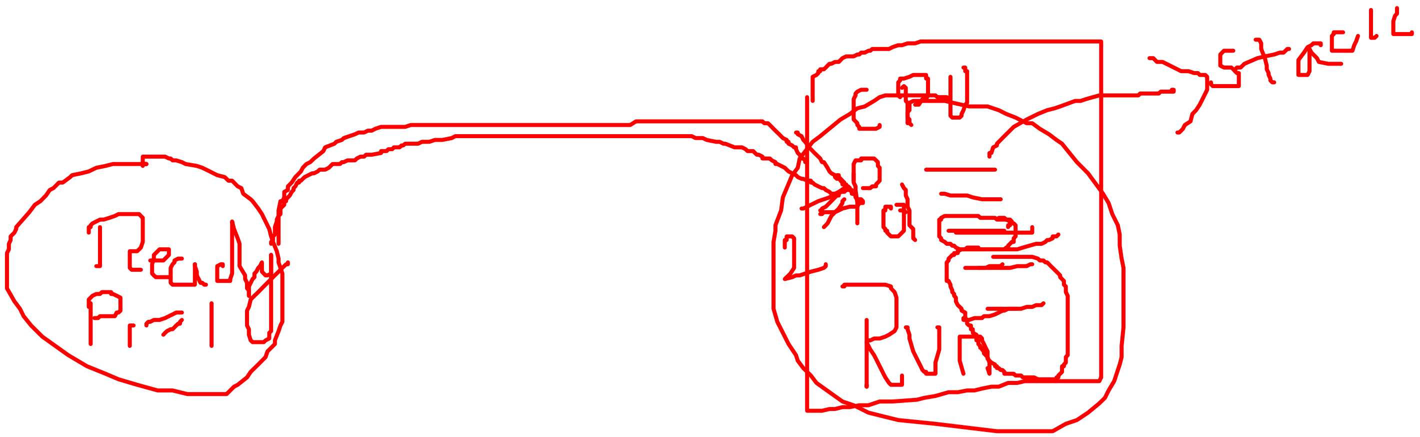


Contd…

- Two possibilities for the address space of the child relative to the parent:
  - ✓ The child may be an exact duplicate of the parent
  - ✓ The child process may have a new program loaded into its address space

Operations on Processes – Process Preemption and Blocking

- Preemption : temporarily interrupting the running process
- Process blocking: Running process goes for an I/O



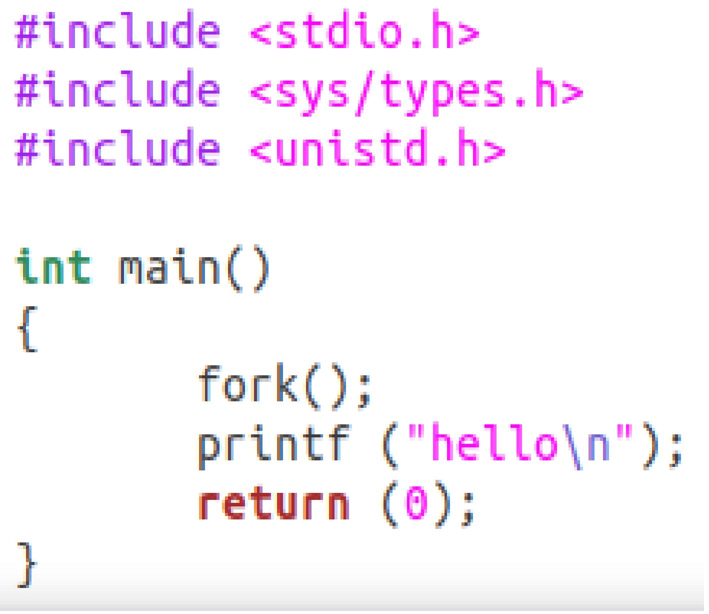
Operations on Processes

- Processes may request their own termination by making the exit( ) system call
- When a process terminates, all of its system resources are freed up, open files flushed and closed, etc.
- The process termination status and execution times are returned to the parent
- **Orphan process**
- **Zombie process.**

fork() system call

- To create new process
- Parent process : process which invoked fork()
- A child process uses the same pc(program counter), same CPU registers, same open files which are used in the parent process.
- fork() takes no parameters and returns an integer value
  - ***Negative Value***: creation of a child process was unsuccessful.
  - ***Zero***: Returned to the newly created child process.
  - ***Positive value***: Returned to parent or caller. The value contains process ID of newly created child process.

Example 1

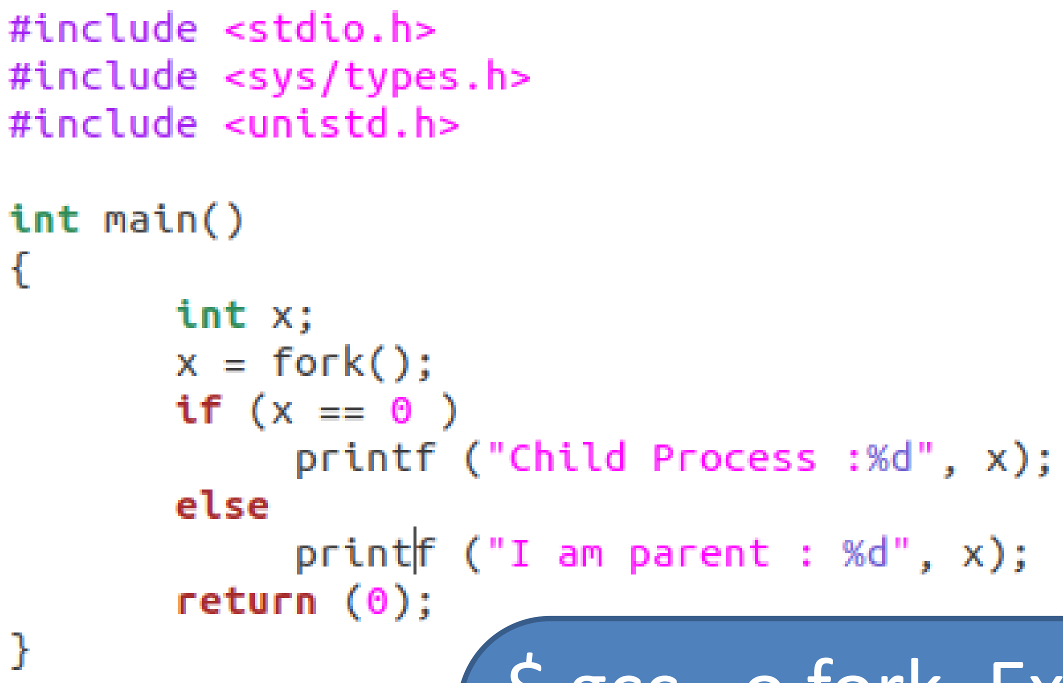


```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
        fork();
        printf ("hello\n");
        return (0);
}
```

```
$ gedit fork.c
$ gcc –o fork_EX1 fork_Ex1.c
$ ./fork_Ex1
hello
hello
$
```

Example 2

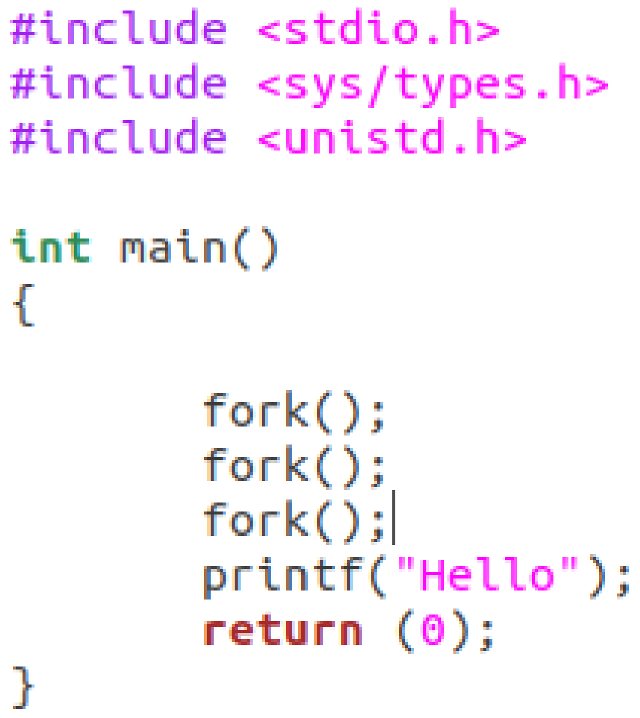


```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
        int x;
        x = fork();
        if (x == 0 )
             printf ("Child Process :%d", x);
        else
             printf ("I am parent : %d", x);
        return (0);
}
```

```
$ gcc –o fork_Ex3 fork_Ex3.c
$ ./fork_Ex3
Child Process : 0
I am Parent : 1234
$
```

Example 3



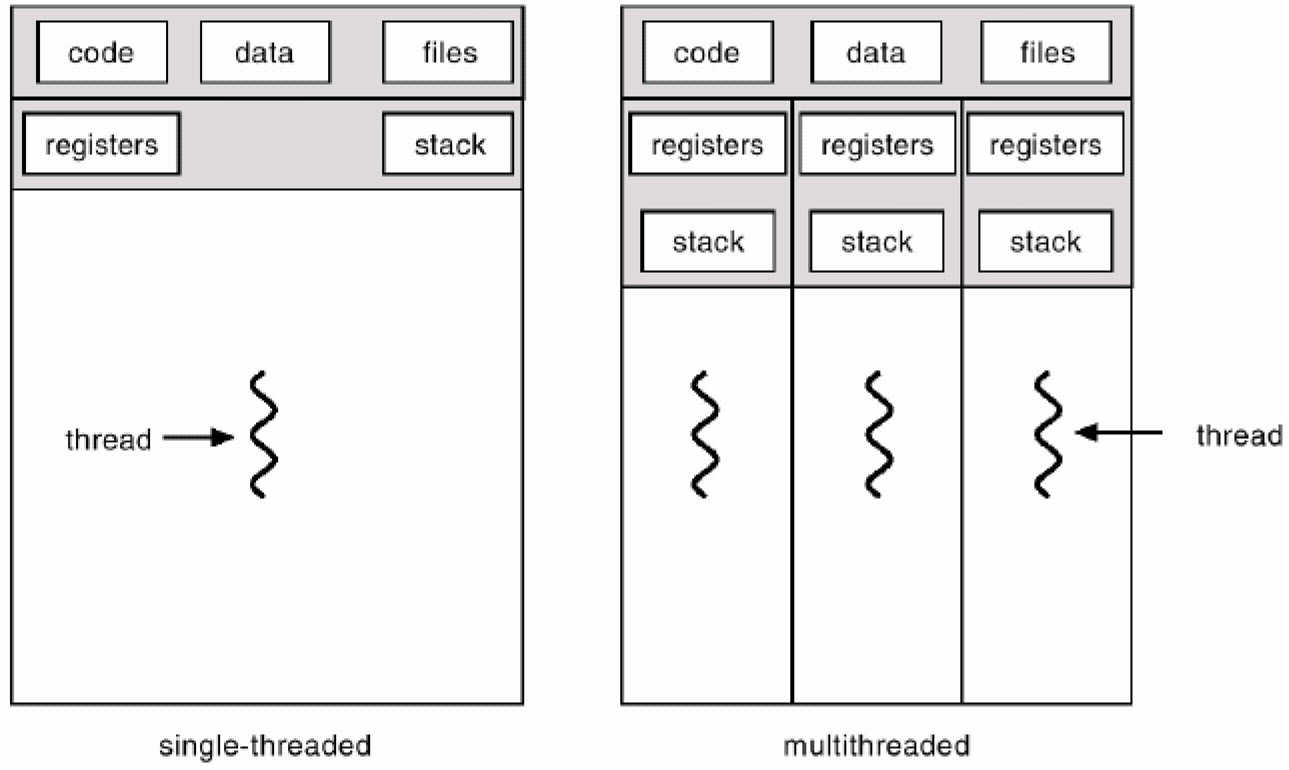
```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{

        fork();
        fork();
        fork();
        printf("Hello");
        return (0);
}
```

Threads

- A thread is a **single sequence stream** within in a process
- A thread has a **program counter (PC), a register set, and a stack space.**



Process Vs. Threads

Process

1. Considered Heavy weight
2. Process creation is costly in terms of resources
3. Program executing as process are relatively slow
4. Process cannot access the memory area belonging to another process
5. Process switching is time consuming
6. One process can contain several threads

Threads

1. Considered light weight
2. Thread creation is very economical
3. Program executing as threads are comparatively faster
4. Thread can access the memory area belonging to another thread within same process
5. Thread switching is faster
6. One thread can belong to exactly one process

Process vs. Thread

Similarities

- Like processes threads share CPU and only one thread active (running) at a time.
- Like processes, threads within a processes, execute sequentially.
- Like processes, thread can create children.
- And like process, only when one thread is blocked, another thread can run.

Process vs. Thread contd.

Differences

- Unlike processes, threads are not independent of one another.
- Unlike processes, all threads can access every address in the task .
- Unlike processes, threads are created to assist one other.

Types of Threads

Threads are implemented in following two ways

- User Level Threads -- User managed threads.
- Kernel Level Threads -- Operating System managed threads.

BITS Pilani
Pilani Campus

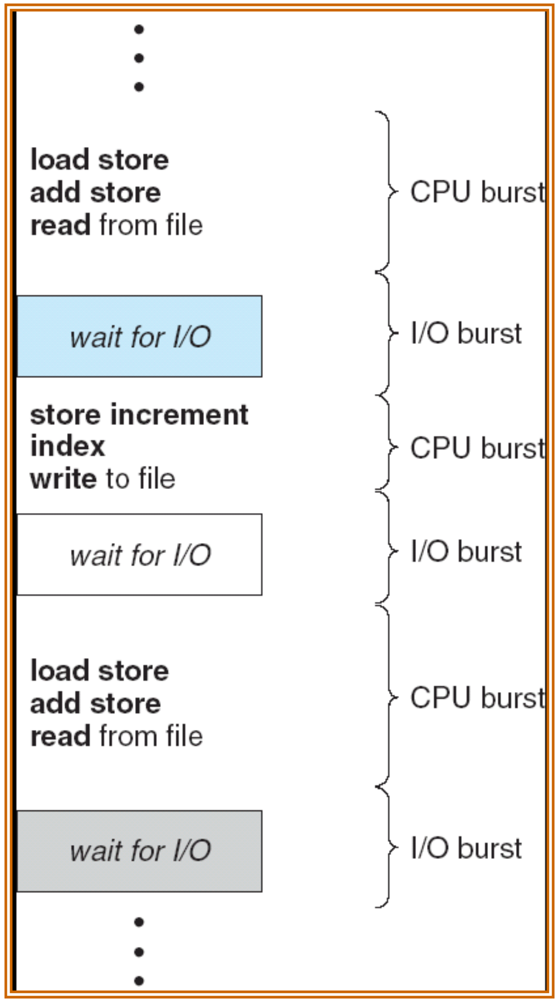
Process Scheduling

Scheduling: Basic Concepts

- **Scheduling**: The assignment of CPU to a process
- **Preemption**: Removal of CPU control from a process
- **Context Switch:** Switching the CPU to another process
    - Save the state of old process
    - Load the saved state of new process
- **PCB** is used to save the information /context of a process

Scheduling: Basic Concepts…

- Maximum CPU utilization obtained with multiprogramming
- Process execution consists of a *cycle* of CPU execution and I/O wait
  - CPU–I/O Burst Cycle
- CPU burst distribution



Process scheduling

- Job queue – set of all processes in the system
- Ready queue – set of all processes residing in main memory, ready and waiting to execute
- Device queues – set of processes waiting for an I/O device
- Processes migrate among the various queues

Types of Scheduler

- Long-term scheduler
- Medium-term scheduler
- Short-term scheduler
- I/O scheduler

Long Term Schedulers

- Also known as Job Scheduler
- Decides which processes should be brought into the ready queue.
- The long-term scheduler controls the **degree of multiprogramming**
    - Main Goal : to have good mix of CPU bound and I/O bound processes
- Executes infrequently, maybe only when process leaves system
- Does not exist on most modern timesharing systems

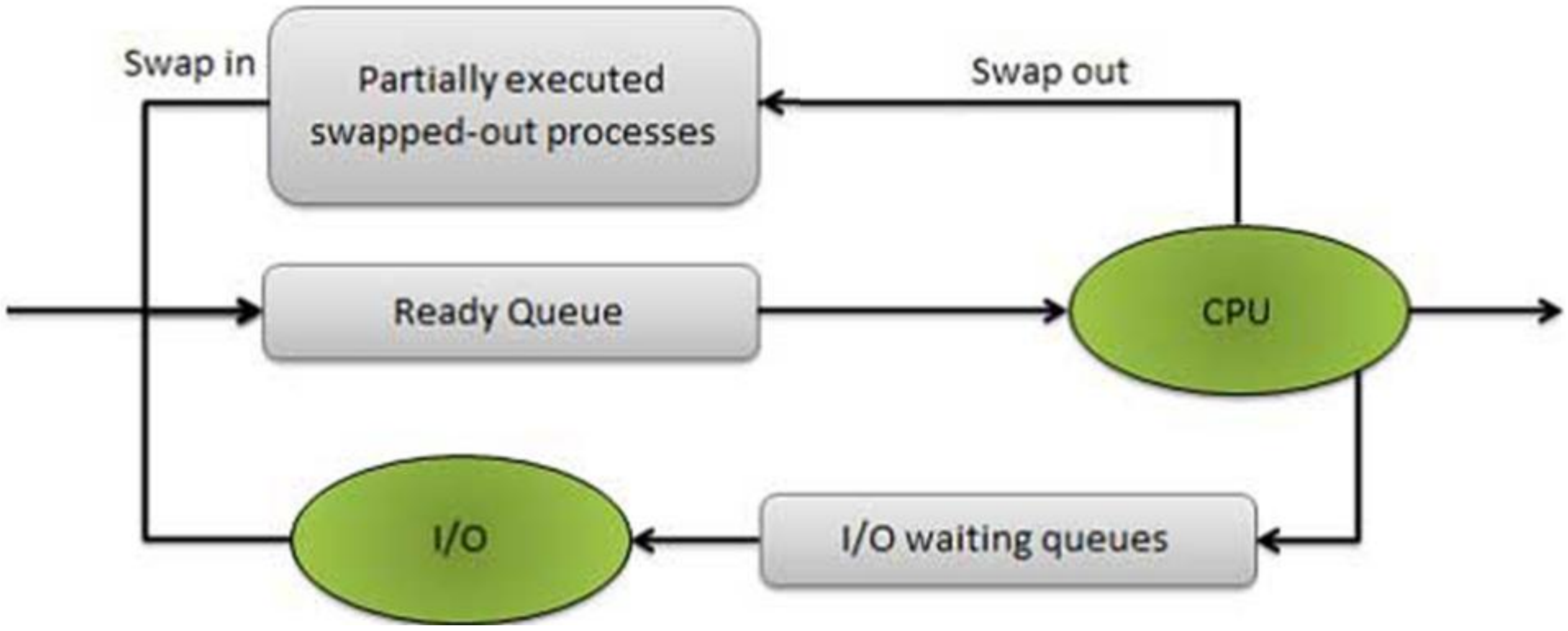
Short Term Scheduler

- Also known as CPU Scheduler or dispatcher
- selects from among the processes that are ready to execute and allocates the CPU to one of them
- Executes frequently
- Runs whenever
  - Process is created and brought in to ready queue
  - Process is terminated
  - Process switches from running to blocked (wait)
  - When interrupt occurs

Short - Term Scheduler…

- Main Goals:
  - Minimize response time
  - Maximize throughput
    - Minimize overhead such as OS overhead, context switching etc.
    - Efficient use of resources
  - Fairness – Share CPU and other resources in an equitable fashion

Addition of Medium Term Scheduling



Contd...

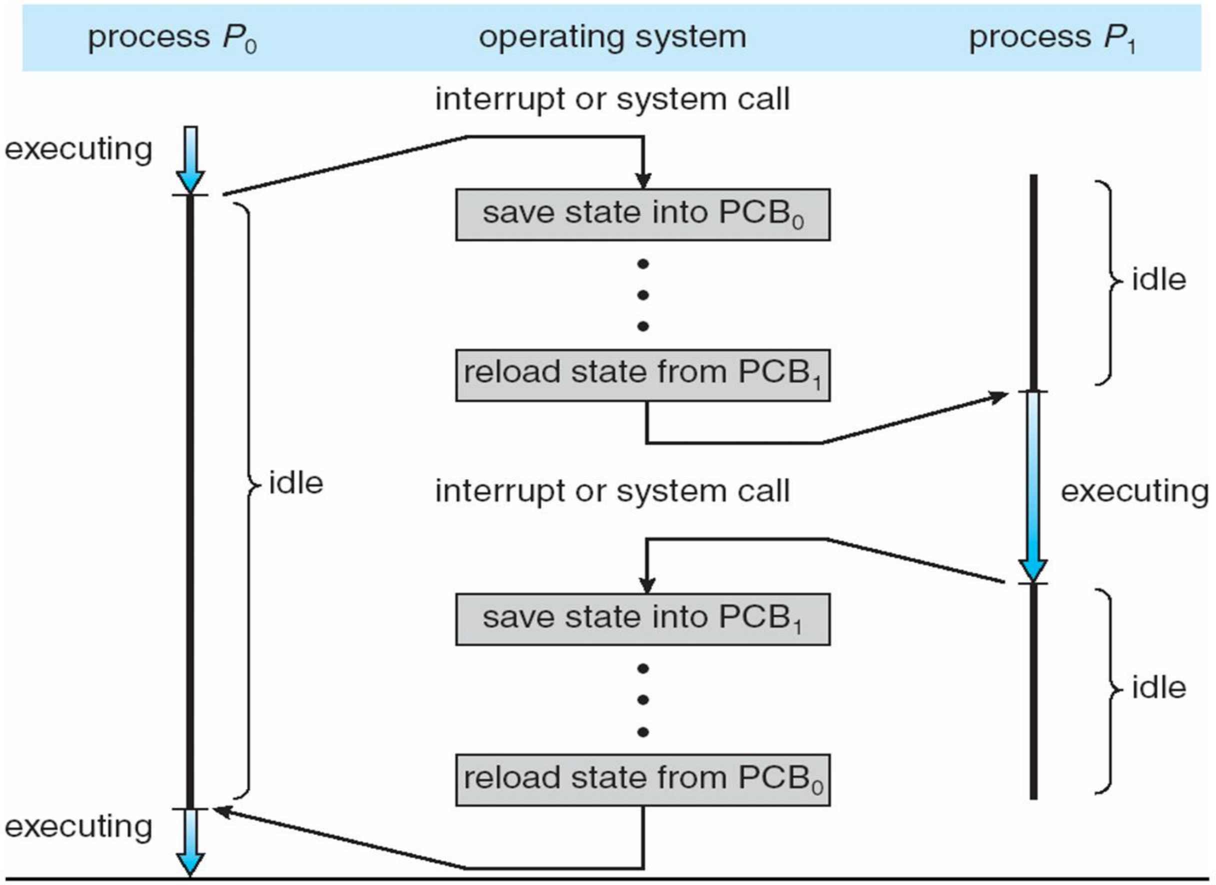
The mid-term scheduler may decide to swap out

1. a process which has not been active for some time
2. a process which has a low priority
3. a process which is page faulting frequently
4. a process which is taking up a large amount of memory in order to free up main memory for other processes, swapping the process back in later when more memory is available,
5. when the process has been unblocked and is no longer waiting for a resource.

I/O Scheduling

The decision as to which process's pending I/O request shall be handled by an available I/O device innovate achieve lead

CPU Switch From Process to Process



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