## Process Management

The Process Concept

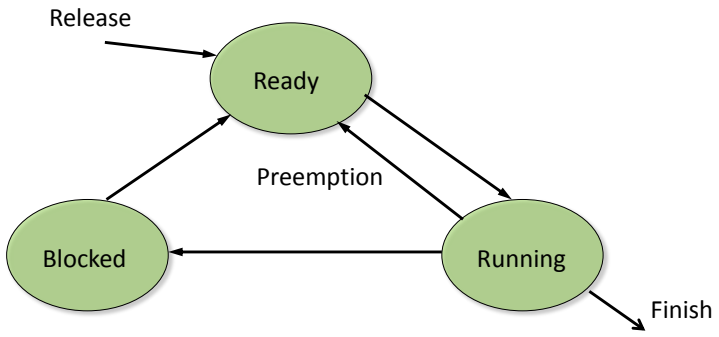
A process is usually defined as a “program in execution”.

A *program* is a static sequence of instructions stored in memory. The *process* is the dynamic operation of executing a program

The process is the fundamental unit of computation that the system must manage. It is the unit to which resources are assigned.

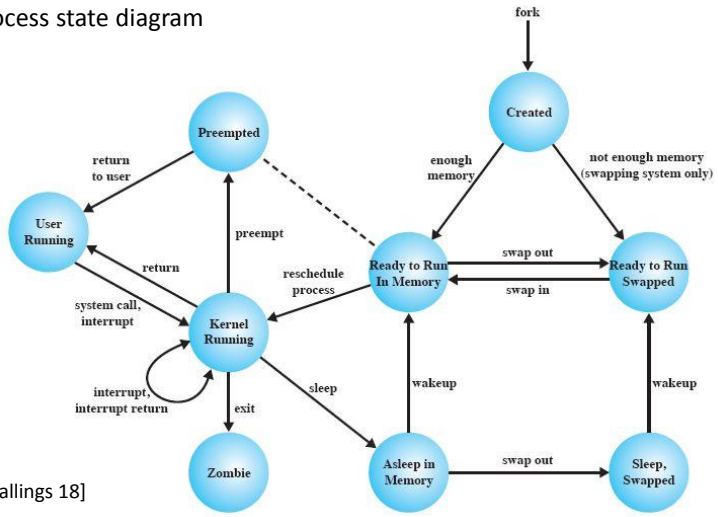


## Process State Diagram



## Process State Diagram

UNIX process state diagram




Source:[Stallings 18]

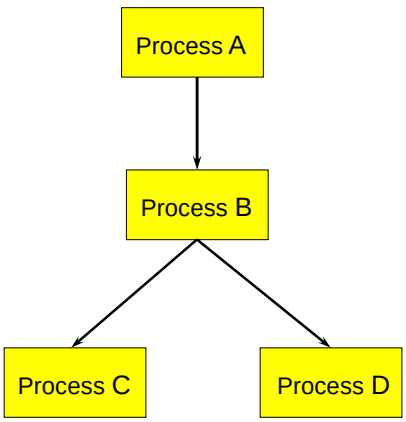


## Process Tree

New process is generated using a special system call.

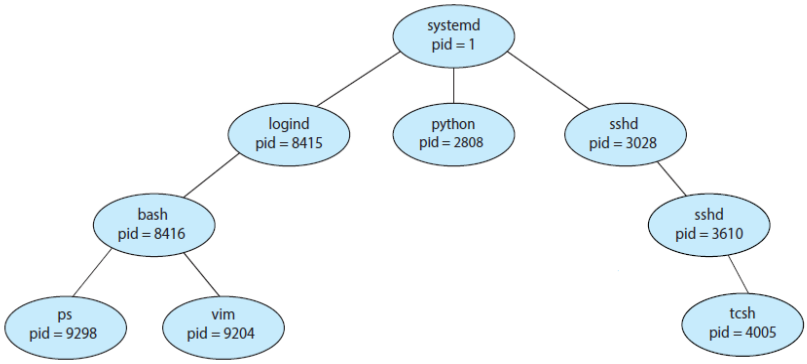
Parent process can spawn one or more child processes. Parent either wait for the child or run concurrently with it.

In some simple embedded systems, all processes are statically started at initialization.



## Process Tree

Example: Typical LINUX process tree




Source: [Silberschatz 18]



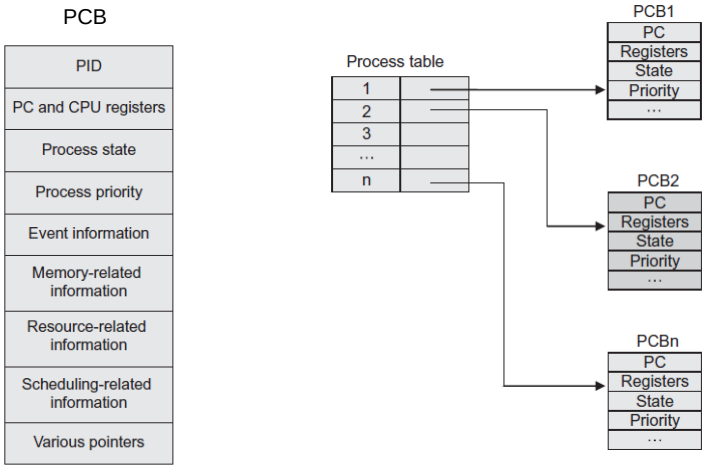
## Process Control Block

Information needed by the system to control a process is stored in an appropriate data structure.

Stored information include:

- Process id, parent id, owner id.
- Process context.
- Process state.
- Memory and I/O resources assigned to the process.

## Process Control Block






## Process Creation in POSIX

POSIX (Portable Operating System Interface) is a family of IEEE standards (1003.1, 1003.1b for RT extensions) that aim at maintaining compatibility between operating systems. POSIX defines the application programming interface (API) rather than implementation itself.

Applications can be migrated between POSIX-compliant operating systems with little modifications.

Process generation in POSIX is based on the *fork* system call. *fork* is used to create a new process by generating an exact duplicate of the calling process. It then returns 0 to the child process, and the child's process id to the parent process. If some error occurs, fork returns -1.

## Process Creation in POSIX

```
pid_t pid = fork();

   if (pid == -1) printf("Error while forking \n");

   else if (pid == 0) {
        exec function .....
                        }
   else  {
        wait function .....
         }
```



## Multithreading

So far, we assumed that a process has a single *thread* of execution: i.e. a single sequence of executed instructions.

In many applications, however, several parts of the same process can run in parallel. These parts share memory address space and other resources.

In a *multithreading* system, a process may be composed of several threads that run in parallel.

Memory space, I/O devices, ..etc. are assigned to a process. However, CPU time is assigned to a single thread. Each thread has independent state and context.

## Multithreading

Advantages of Multithreading

- Process can remain responsive if a part of it is blocked or performs a long operation.
- Creating a new thread of an existing process requires less time and resources than spawning a new process. Also context switching between threads of the same process is faster than switching to another process.
- In a mutliprocessor system, threads can be assigned to different processors (or cores), thus speeding up the execution of a given process.



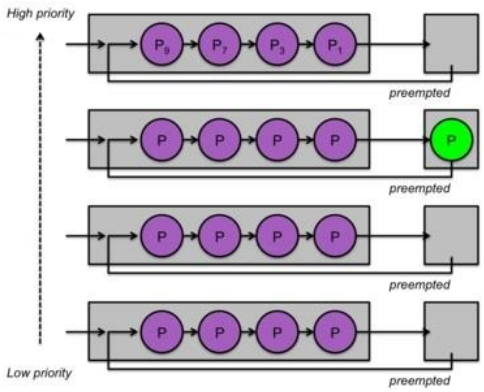
## Process Scheduling

Scheduling algorithm determines the order in which tasks are executed on the processor(s).

A set of tasks is said to be schedulable with an algorithm A if A generates an execution schedule that satisfies all constraints on the tasks in the set.

In multitasking non-RT systems, scheduling is typically done by round-robin scheduling with multiple queues corresponding to different priority levels. Priority levels are allowed to be adjusted based on the recent process behavior.

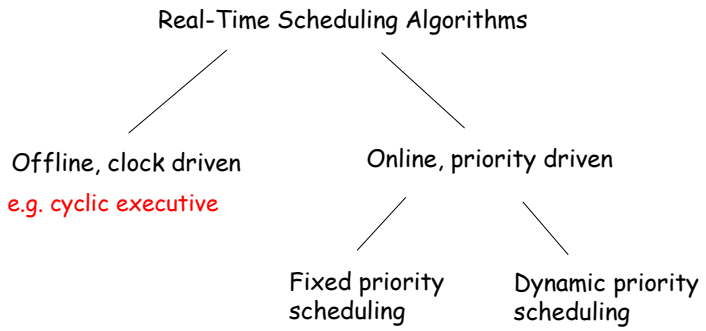
## Process Scheduling




For Real-time systems, especially hard RT systems, this is not suitable.



## Real-time Scheduling



## Cyclic Executive Method

A widely used method to implement real-time programs, especially in embedded systems.

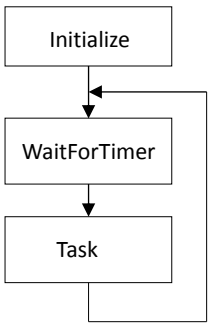
Tasks are run in predefined and predictable manner. It can hence be used to ensure that time constraints are met.

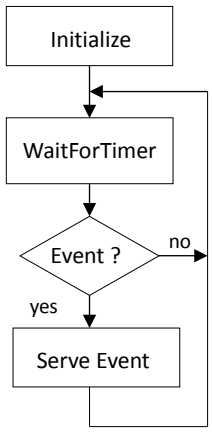
This method can provide an inexpensive alternative to the use of an RTOS.



## Cyclic Executive Method

We will assume that system runs periodic tasks (either periodic execution of task or periodic polling for events).

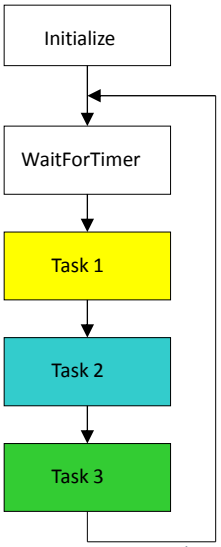


## Cyclic Executive Method

To run multiple tasks, program may be implemented as shown.

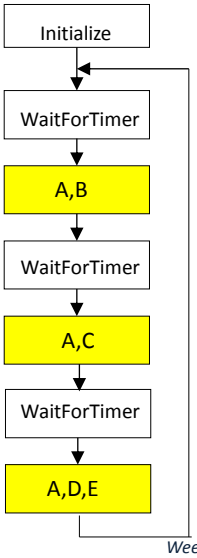
However, different tasks may have different periods and deadlines.






## Cyclic Executive Method

Cyclic executive approach interleaves the operation of several tasks so that all periods and deadlines are satisfied.



## Cyclic Executive Method

○ System will have a major cycle which is continuously repeated. Each task will be executed at least once in each major cycle.

The major cycle length is thus taken as the least common multiple of task periods.

○ Major cycle is divided into a number of minor cycles or frames. A number of tasks will be called in each frame, with each task executed at most once. We take $\Phi_i = 0$ for all tasks.

Frame boundaries correspond to the instants at which correct timing is enforced. Exact timing cannot be controlled inside a frame. For simplicity, we assume equal length frames.



## Cyclic Executive Method

Example (1):

| Task | Period | Execution time | Deadline |
|---|---|---|---|
| A | 25 | 10 | 25 |
| B | 25 | 8 | 25 |
| C | 50 | 5 | 50 |
| D | 50 | 4 | 50 |
| E | 100 | 2 | 100 |

Frame 1 (25 ms): A, B, C

Frame 2 (25 ms): A, B, D, E

Frame 3 (25 ms): A, B, C

Frame 4 (25 ms): A, B, D

Major Cycle
100 ms

## Cyclic Executive Method

```
(* loop *)
    if (next_minor_cycle_start_time) (* from rtc interrupt *)
    then
        next_minor_cycle_start_time := false;
        case (minor_cycle) of
            1: call P1_handler; call P2_handler;
            2: call P1_handler; call P3_handler;
            3: call P1_handler; call P2_handler;
            4: call P1_handler;
        end_case;
        if (minor_cycle < last_minor_cycle)
        then minor_cycle := minor_cycle + 1;   (*    next minor cycle *)
        else minor_cycle := first_minor_cycle;  (* repeat major cycle *)
        end_if;
        if (next_minor_cycle_start_time)
        then … ; (* job overrun *)
        end_if;
    end_if;
(* end_loop *)
```



## Constraints on Frame Length f

- It must divide the major cycle length.
- It must be greater than or equal to the WCET of any task.

$$f \geq p_i \qquad i = 1,2,\ldots,n$$

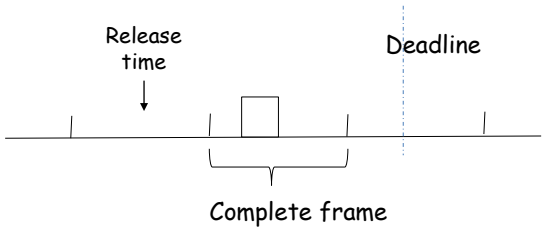
- It must be less than or equal to the deadline of any task.

$$f \leq d_i \qquad i = 1,2,\ldots,n$$

- At least one complete frame should exist between a task release time and the corresponding deadline.

$$2f - \gcd(f, T_i) \leq d_i \qquad i = 1,2,\ldots,n$$

## Constraints on Frame Length f




If task is executed in a complete frame between release time and deadline, there is no need for timing checks within frame. (We are sure that task is executed after release time and before deadline). Otherwise, timing checks within frame will be necessary.

Note that satisfaction of constraints on frame length does not guarantee that a valid schedule can be constructed.



## Constraints on Frame Length f

Example(2): Three tasks with periods of 14, 20, and 22 ms and execution times of 1,2, and 3 ms respectively. For all tasks, deadline is equal to the period.

What are the possible values for the frame?

- After executing tasks within a frame, system remains idle, or some background tasks may be executed.

- To reduce response time to sporadic events, corresponding tasks are typically executed before periodic tasks within a frame.

## Constraints on Frame Length f

To satisfy constraints it is often necessary to break up the task into a number of subtasks, each fitting within a frame. Execution of task may thus be spread over a number of frames.

One common method for task decomposition is to split it into a number of coroutines with shorter execution time:

$$S \Longrightarrow S_1; S_2; S_3; \ldots\ldots; S_m$$

Note that splitting is done by the programmer and not by system preemption.



## Advantages of Cyclic Executive Method

- Satisfies timing constraints without the need for an RTOS with preemptive scheduling.
- Efficient: less decision and switching overheads.
- Easier to handle concurrency problems since there is no preemption.
- Different cycles can be defined for different modes of operation.

## Disadvantages of Cyclic Executive Method

- ✘ Not flexible and difficult to maintain.
- ✘ Based on trial and error and difficult to optimize.
- ✘ May result in slower response time for high priority processes, compared to preemptive techniques.