Great Place To Work® Certified FEB 2023 – FEB 2024 INDIA ™

UPES

UNIVERSITY OF TOMORROW

nirf RANKING 2023

#**RANKED 52** IN INDIA

#University Category

QS WORLD UNIVERSITY RANKINGS

NO.1 PVT. UNIVERSITY IN ACADEMIC REPUTATION IN INDIA

ACCREDITED WITH GRADE A NAAC

ACCREDITED **GRADE 'A'** BY NAAC

E-LEAD

PERFECT SCORE OF **150/150** AS A TESTAMENT TO EXECEPTIONAL E-LEARNING METHODS

Unit 2 : Process and Thread Management

Lecture 2

Submitted by:

Syed Sajid Hussain

School of Computer Science

UPES, Dehradun

India

UPES
UNIVERSITY OF TOMORROW

Table of Contents

UPES
UNIVERSITY OF TOMORROW

Learning & Course Outcomes

LO1:Explain the objectives and importance of process scheduling in an operating system.

LO2:Understand the concept and mechanisms of context switching.

LO3:Describe various operations on processes, including process creation and termination.

Course Outcomes

- CO2: Evaluate and analyze process and thread scheduling techniques, discerning their benefits and challenges.

UPES
UNIVERSITY OF TOMORROW

Process Scheduling

- The objective of multiprogramming is to have some process always running, to maximize CPU utilization.
- The objective of time sharing is to switch the CPU 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 for program execution on the CPU.

Process scheduling involves three things:

1. Scheduling Queues
2. Schedulers
3. Context Switch

UPES
UNIVERSITY OF TOMORROW

Scheduling Queues

There are several queues are implemented in operating system such as Job Queue, Ready Queue, Device Queue.

- **Job Queue:** It consists of all processes in the system. As processes enter the system, they are put into a **job queue**.
- **Ready Queue:** The processes that are residing in main memory and they are ready and waiting to execute are kept on a list called the **Ready Queue**. Ready queue is generally stored as a linked list. A ready-queue header contains pointers to the first and final PCBs in the list. Each PCB includes a pointer field that points to the next PCB in the ready queue.
- **Device Queue:** Each device has its own device queue. It contains the list of processes waiting for a particular I/O device.

Scheduling Queues(Continuing…)

Consider the given Queuing Diagram:

- Two types of queues are present: the **Ready Queue** and a set of **Device Queues**.
- CPU and I/O are the resources that serve the queues.
- A new process is initially put in the ready queue. It waits there until it is selected for execution or **dispatched**.
- Once the process is allocated the CPU and is executing, one of several events could occur:
    1. The process could issue an I/O request and then be placed in an I/O queue.
    2. The process could create a new child process and wait for the child's termination.
    3. The process could be removed forcibly from the CPU, because of an interrupt and be put back in the ready queue.

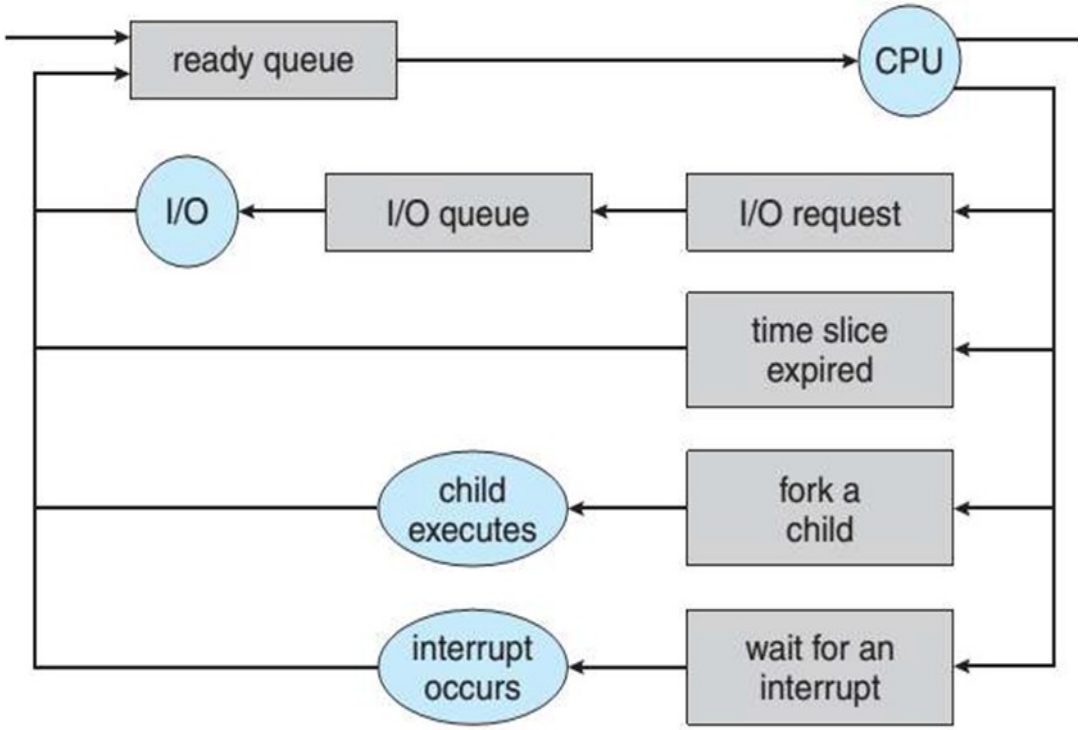


Fig: Queuing diagram representation of process scheduling

Schedulers

A process migrates among the various scheduling queues throughout its lifetime. For scheduling purpose, the operating system must select processes from these queues. The selection process is carried out by the **Scheduler.**

There are three types of Schedulers:

1. Long Term Scheduler
2. Short Term Scheduler
3. Medium Term Scheduler

Long Term Scheduler (New to ready state)

- Initially processes are spooled to a mass-storage device (i.e Hard disk), where they are kept for later execution.
- Long-term scheduler or job scheduler selects processes from this pool and loads them into main memory for execution. (i.e. from Hard disk to Main memory).
- The long-term scheduler executes much less frequently, there may be minutes of time between creation of one new process to another process.
- The long-term scheduler controls the **degree of multiprogramming** (the number of processes in memory).

Schedulers(Continuing…)

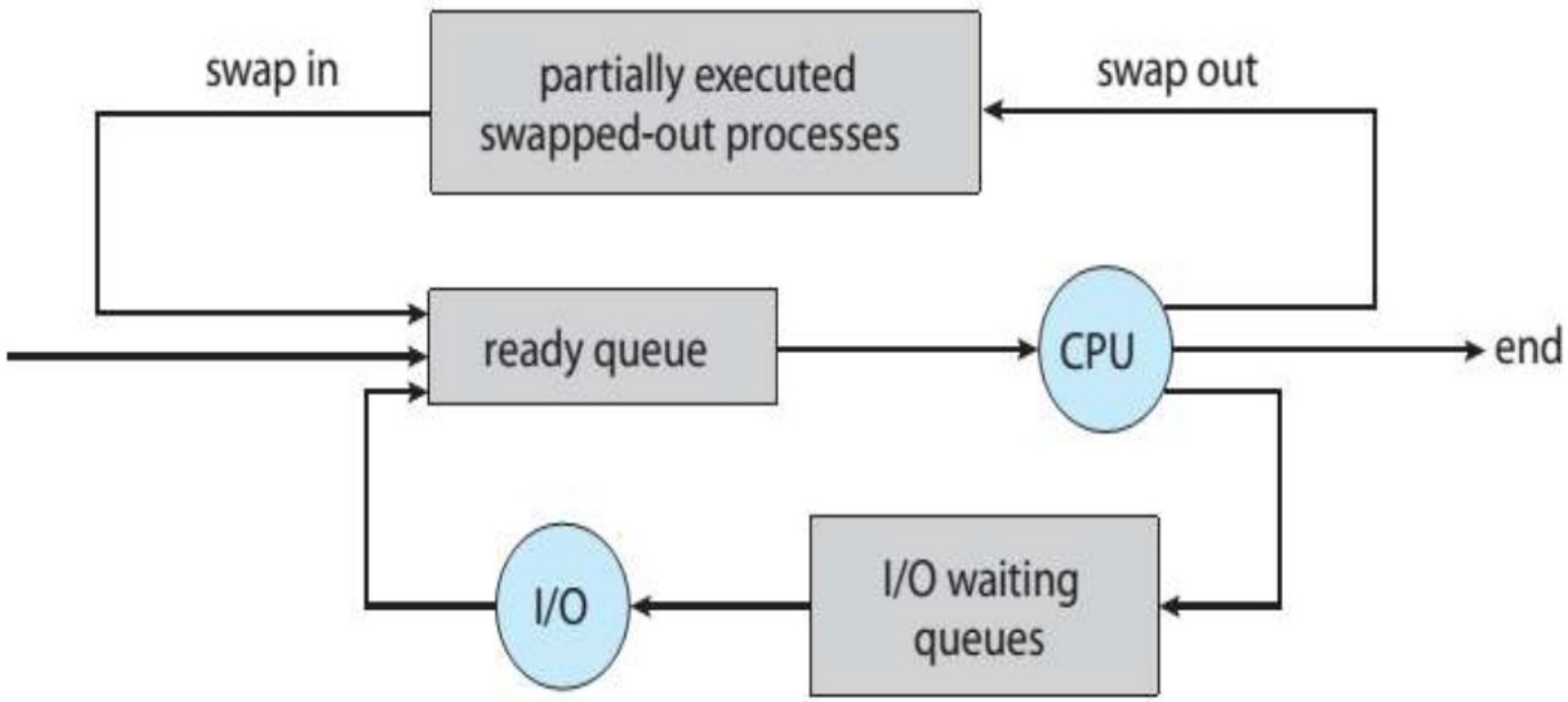


Fig: Process Scheduler

Continuing…

Short Term Scheduler (Ready to Running)

- Short-term scheduler or CPU scheduler selects from among the processes that are ready to execute and allocates the CPU to one of them. (i.e. a process that resides in main memory will be taken by CPU for execution).
- The short-term scheduler must select a new process for the CPU frequently.
- The short term scheduler must be very fast because of the short time between executions of processes.

Medium Term Scheduler

Medium Term Scheduler does two tasks:

1. **Swapping:** Medium-term scheduler removes a process from main memory and stores it into the secondary storage. After some time, the process can be reintroduced into main memory and its execution can be continued where it left off. This procedure is called Swapping.
2. Medium Term Scheduler moves a process from CPU to I/O waiting queue and I/O queue to ready queue.

Continuing…

The processes can be described as two types:

1. I/O bound process is one that spends more of its time doing I/O than it spends doing computations.
2. CPU Bound process using more of its time doing computations and generates I/O requests infrequently.

The long-term scheduler selects a good *process mix* of I/O-bound and CPU-bound processes.

- If all processes are I/O bound, the ready queue will almost always be empty, and the CPU will remain idle for long time because I/O device processing takes a lot of time.
- If all processes are CPU bound, the I/O waiting queue will almost always be empty. I/O devices will be idle, and CPU is busy for most of the time.
- Thus, if the system maintains the combination of CPU bound and I/O bound processes then the system performance will be increased.

Note: Time-sharing systems such as UNIX and Microsoft Windows systems often have no long-term scheduler but simply put every new process in memory for the short-term scheduler.

UPES
UNIVERSITY OF TOMORROW

Context Switching

- Switching the CPU from one process to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a **Context Switch**.
- The context is represented in the PCB of the process. It includes the value of the CPU registers, the process state and memory-management information.
- When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.
- Context-switch time is pure overhead, because the system does no useful work while switching. Context switch time may be in few milliseconds.

Context Switching(Continuing…)

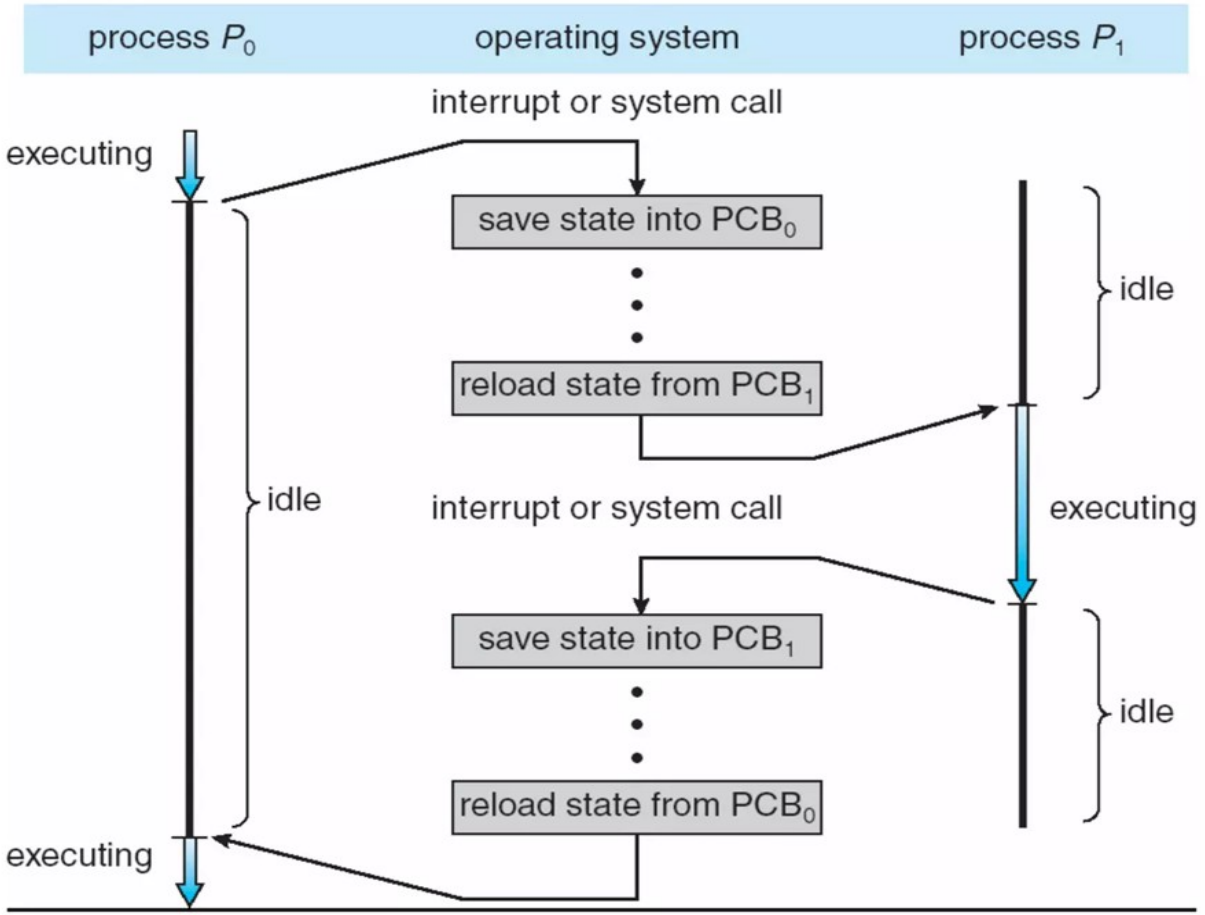


Fig: CPU Switch from Process to Process

Operations on Processes

1. Process Creation
2. Process Termination

Process Creation

- During the execution of a process in its life time, a process may create several new processes.
- The creating process is called a parent process and the new processes are called children process.
- Each of these new processes may create other processes forming a **tree** of processes.
- Operating system identifies processes according to **process identifier** (**pid**).
- Pid provides an unique integer number for each process in the system.
- Pid can be used as an index to access various attributes of a process within the kernel.

Operations on Processes (Continuing…)

The below figure shows the process tree for the Linux OS that shows the name of each process and its process ID (pid) . In Linux process is called task.

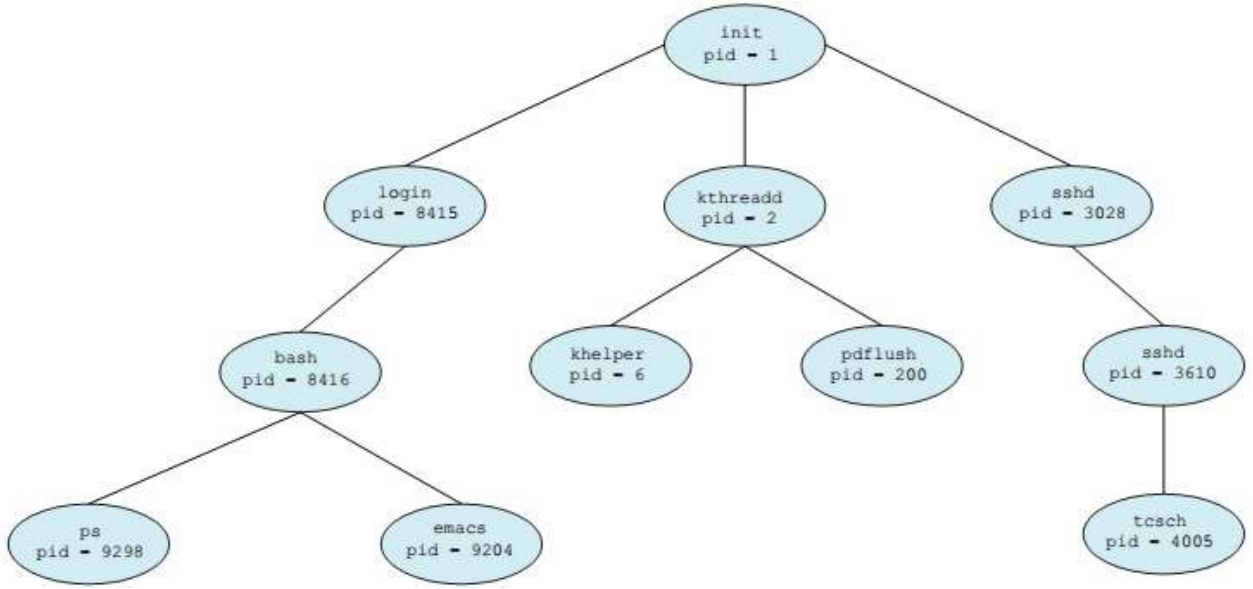


Fig: Process Tree

Continuing…

- The init process always has a pid of 1. The init process serves as the root parent process for all user processes.
- Once the system has booted, the init process can also create various user processes, such as a web or print server, an ssh server etc.
- *kthreadd* and *sshd* are child processes of init.
- The *kthreadd* process is responsible for creating additional processes that perform tasks on behalf of the kernel.
- The *sshd* process is responsible for managing clients that connect to the system by using secure shell (ssh).
- **ps** command is used to obtain a list of processes:

ps –el

The command will list complete information for all processes currently active in the system.

- When a process creates a child process, that child process will need certain resources such as CPU time, memory, files, I/O devices to accomplish its task.
- A child process may be able to obtain its resources directly from the operating system or it may be constrained to a subset of the resources of the parent process.

UPES
UNIVERSITY OF TOMORROW

Continuing…

- The parent may have to partition its resources among its children or it may be able to share some resources such as memory or files among several of its children.
- Restricting a child process to a subset of the parent's resources prevents any process from overloading the system by creating too many child processes.

When a process creates a new process there exist two possibilities for execution:

1. The parent continues to execute concurrently with its children.
2. The parent waits until some or all its children have terminated.

There are also two address-space possibilities for the new process:

3. The child process is a duplicate of the parent process (i.e) it has the same program and data as the parent.
4. The child process has a new program loaded into it.

Continuing…

Process System calls in Unix/ Linux: fork( ), exec( ), wait( ), exit( )

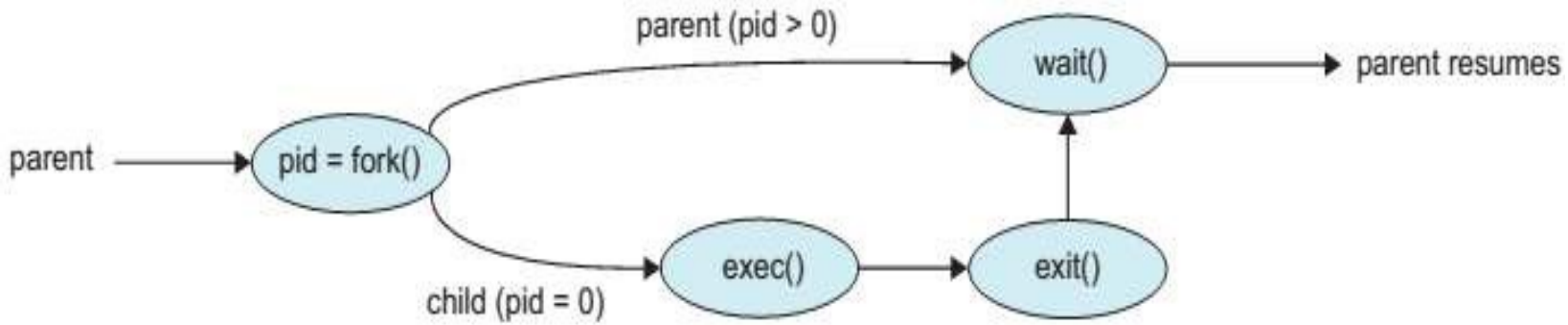
- **fork( )**: In UNIX OS a new process is created by the **fork( )** system call.
- The new process consists of a copy of the address space of the original process. This mechanism allows the parent process to communicate easily with its child process.
- Both the parent and the child processes continue execution at the instruction after the fork( ).
- For the new child process (i.e. Child Process) the return code for the fork( ) is zero.
- The nonzero process identifier of the child is returned to the parent.

- **exec( )**: After a fork( ) system call, one of the two processes typically uses the **exec( )** system call to replace the process's memory space with a new program.
- The exec( ) system call loads a binary file into memory and starts its execution.
- In this way, the two processes are able to communicate and then go their separate ways.

UPES
UNIVERSITY OF TOMORROW

Continuing…

- **wait( ):** The parent can create more children or if the parent has nothing else to do while the child process is running then the parent process can issue a **wait( )** system call to move itself out of the Ready Queue until the child process terminates
- The call to exec( ) overlays the process's address space with a new program or the call to exec( ) does not return control unless an error occurs.



Continuing…

Program for 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;
}
```

Continuing…

The C program in the previous slide shows the UNIX system calls fork, exec, wait. Two different processes
are running copies of the same program.

- The only difference is that the value of pid for the child process is zero, while the value of pid for the parent is an integer value greater than zero (i.e. the actual pid of the child process).
- The child process inherits privileges and scheduling attributes from the parent, as well as certain resources such as open files.
- The child process then overlays its address space with the UNIX command /bin/ls (used to get a directory listing) using the execlp( ) system call (execlp( ) is a version of the exec( ) system call).
- The parent waits for the child process to complete with the wait( ) system call.
- When the child process completes by either implicitly or explicitly invoking exit( ), the parent process resumes from the call to wait( ), where it completes using the exit( ) system call.

Continuing…

Process Termination: exit( )

- A process terminates when it finishes executing its final statement and asks the operatingsystem to delete it by using the **exit( )** system call.
- The process may return a status value to its parent process via the wait( ) system call.
- All the resources of the process including physical and virtual memory, open files and I/O buffers are deallocated by the operating system.

A parent may terminate the execution of one of its children for a variety of reasons such as:

1. The child has exceeded its usage of some of the resources that it has been allocated.
2. The task assigned to the child is no longer required.
3. The parent is exiting and the operating system does not allow a child to continue if its parent terminates.

Operations on Processes

Cascading Termination

- If a parent process terminates either normally or abnormally then all its children must also be terminated is referred as Cascading Termination. It is normally initiated by operating system.

- In Linux and UNIX systems, a process can be terminated by using the *exit( )* system call providing an exit status as a parameter:

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

- Under normal termination, exit( ) may be called either directly (i.e. exit(1)) or indirectly (i.e. by a return statement in main( )).
- A parent process may wait for the termination of a child process by using the wait( ) system call.
- The wait( ) system call is passed a parameter that allows the parent to obtain the exit status of the child. This system call also returns the process identifier of the terminated child so that the parent can tell which of its children has terminated:

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

Continuing…

Zombie process

A process that has terminated but whose parent has not yet called wait( ) is known as a **zombie** process.

- When a process terminates, its resources are deallocated by the operating system. Its entry in the process table must remain there until the parent calls wait( ), because the process table contains the process's exit status.
- Once the parent calls wait( ), the process identifier of the zombie process and its entry in the process table are released.

Orphan Processes

If a parent did not invoke wait( ) and instead terminated, thereby leaving its child processes as **orphans** are called Orphan processes.

- Linux and UNIX address this scenario by assigning the init process as the new parent to orphan processes.
- The init process periodically invokes wait( ), thereby allowing the exit status of any orphaned process to be collected and releasing the orphan's process identifier and process-table entry.

MCQ's

1. What is the primary function of the Process Control Block (PCB)?
a) Manages memory allocation for the process
b) Stores the context of a process
c) Schedules processes
d) Handles I/O operations

2. Which process scheduling algorithm assigns the highest priority to the process with the shortest burst time?
a) First-Come, First-Served (FCFS)
b) Shortest Job First (SJF)
c) Round Robin (RR)
d) Priority Scheduling

MCQ's

3.In a multilevel queue scheduling algorithm, processes are:
a) Divided into several queues, each with its own scheduling algorithm
b) Scheduled based on their priority
c) Executed in a round-robin manner
d) All of the above

4.What is the main advantage of threads over processes?
a) Threads are easier to create
b) Threads share the same memory space
c) Threads require less CPU time to switch
d) All of the above

MCQ's

5. Which scheduling scope is concerned with how threads are scheduled within a single process?
a) System scope
b) Content scope
c) Process scope
d) User scope

6. In the many-to-many threading model:
a) Many user-level threads are mapped to many kernel threads
b) Each user-level thread is mapped to a single kernel thread
c) One user-level thread is mapped to many kernel threads
d) Many user-level threads are mapped to a single kernel thread

Answers

1. b) Stores the context of a process
2. b) Shortest Job First (SJF)
3. a) Divided into several queues, each with its own scheduling algorithm
4. d) All of the above
5. b) Content scope
6. a) Many user-level threads are mapped to many kernel threads

Summary/Key Points

- Process Concept: Basic unit of execution, includes program code and activity.
- Process States: New, Running, Waiting, Ready, Terminated.
- Process Control Block (PCB): Stores state, program counter, CPU registers, memory management info, accounting, and I/O status.
- Schedulers: Long-Term (loads processes into memory), Short-Term (allocates CPU to ready processes), Medium-Term (manages degree of multiprogramming).
- Scheduling Queues: Job, Ready, Device.

References

1. D. M. Dhamdhere, Operating Systems - A Concept Based Approach, Third Edition, Tata McGraw-Hill, 2012
2. Avi Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Concepts, John Wiley & Sons, Ninth edition, 2012
3. M.J.Bach, The design of UNIX operating system, Prentice Hall of India, 1986
4. Uresh Vahalia, UNIX Internals : The New Frontiers, Pearson Education India, 2008
5. A.S.Tanenbaum, Operating System : Design and Implementation,Prentice Hall of India, Third edition, 2009

UPES
UNIVERSITY OF TOMORROW

Coming up-next lecture

1. Process Scheduling
2. Process Scheduling Algorithms

Thank You

UPES
UNIVERSITY OF TOMORROW