

MIT WORLD PEACE UNIVERSITY

Computer Networks
Second Year B.Tech Semester 3
Academic Year 2022-23

OPERATING SYSTEMS

NOTES FROM TANANBAUM AND CLASSES

Prepared By

P34. Krishnaraj Thadesar

Batch A2

August 30, 2022

Contents

1 Processes	2
1.1 Process states	2
1.1.1 Suspended State	2
2 Process Control Block	2
3 Switches	3
3.1 Context Switches	3
4 Process Execution	3
5 Process Creation	3
5.1 fork()	4
6 Process Termination	5
7 Threads	5
7.1 Differences between thread and processes	5
7.2 User Level Threads	6
8 Scheduling	6
8.1 CPU or Process Scheduling	7
8.2 Scheduling Algorithms	7
8.3 Types of schedulers	8
9 Scheduling Algorithms	8
9.1 First Come First Serve FSFC	8
9.2 Shortest Job First Scheduling	9

1 Processes

A process is an instance of a program in execution. It is an entity that can be assigned to and executed on a processor.

1. Process is comprised of Program Code
2. Data
3. Stack
4. A number of attributes describing the state of process.

1.1 Process states

1. New
2. Ready
3. Running
4. Waiting
5. Terminated
6. Suspended

1.1.1 Suspended State

- Process is faster than IO so many processes could be waiting for IO
- Swap this process to disk (SSD/ HDD) to free up RAM memory.
- Ready or waiting state becomes suspended state when swapped to disk.

2 Process Control Block

It is a data structure maintained by the Operating System. It holds all necessary information related to Process. Information associated with each process is as follows:

1. Process state
2. Program Counter
3. CPU Registers
4. CPU Scheduling information
5. Memory Management Information
6. Accounting Information
7. IO Status information

3 Switches

3.1 Context Switches

1. It switches the execution of a process to another, so for that it has to do some stuff,
2. It saves the state of the first program, and then reloads the state of the next one.
3. And only then it runs the next process. This takes time, and is a major disadvantage.
4. It is a mode switch, but a mode switch isn't a context switch.
5. It is a mode switch coz it requires you to switch mode from user to kernel.

4 Process Execution

Consider three processes being executed, all are in the meory, plus the dispatcher.

Dispatcher Dispatcher is a small program which switches from one program to another. -

5 Process Creation

When a new process is created, the following happens:

1. Allocates space to the process in memory
2. Assign a unique Process ID to the Process
3. A process control Block PCB gets associated with the process
4. OS Maintains pointers to each process's PCB in a process table sothat it can access the PCB quickly.

Reasons to create a Process

1. New User Job
2. Created by OS to provide a service
3. Spawned by existing Process: The action of creating a new process at the explicit request of another process is called process spawning.

After Creation

1. Stay in the parent Process
2. Transfer Control to the child process. The system call for that is called Fork. This child process inherits everything from the parent.
3. Transfer control to another process.

5.1 fork()

A system call fork() is used to create processes. It takes no arguments and returns a process ID. The syntax for the fork system call `pid = fork();`

in the Parent process, pid is the child process

In the child process, pid is 0

1. It allocates a slot in the process table for the new process
2. It assigns a unique ID number to the child process
3. It makes a copy of the context of the parent process.
4. It returns the ID number of the child to the parent process, and a 0 value to the child process process is assigned.
5. It doesn't take any arguments
6. Purpose of fork is to create a new process, which becomes the child process of the caller.
7. After a process is created, both processes will execute the next instruction following the fork system call.
8. To distinguish the parent from the child, the returned value of fork can be used.
9.
 - fork() returns a negative value to the parent if the creation of the child process wasn't successful
 - 0 to the child process if successful, and the PID of thus generated child process to the parent process.
10. Returned process id is of type PID defined in `sys/types.h`
11. Process can use function `getpid()` to retrieve the process ID assigned to this process.
12. Linux would make an exact copy of the parent's address space and give it to the child. Therefore the parent and child process will always have a separate address space.

The OS will make two identical copies of address spaces for parent and child processes. So the parent and child processes have different address spaces. A local variable is:

1. Declared inside the process
2. Created when the process starts
3. Lost when the process terminates

A global variable is:

1. Declared outside the process
2. Created as the process starts
3. Lost when the program ends

The process ID, i.e., PID of the child process created, is returned to the parent process. (In case of failure, -1 is returned to the parent process.)

Zero is returned to the child process. (If it fails, the child process is not created.) If a child process exits at that instant or is interrupted, a signal SIGCHLD is sent to the parent process.

Both parent and child processes independently execute the subsequent commands after the fork() system call.

6 Process Termination

- All the resources held by process are released.
- All the information held in all data structures is removed.
- A process goes back to becoming a program and is stored on the secondary memory.

7 Threads

- A thread is a part of a program.
- It is an execution unit within a process
- All threads of the same process share the same address space.
- All threads have separate stacks and individual Thread IDs.
- Thread is a lightweight process because: The context switching between threads is inexpensive in terms of memory and resources.
- Even if 2 processes are communicating with each other, it is their threads that are communicating.
- They share the same process things like data, open files, descriptors, signals, current working directory, user and group id.
- Each thread has a unique thread ID, which has a set of registers, stack for local variables, return addresses, priorities.

There are 2 types of threads. This is similar to how there is always this binary existence of many basic OS operations and functions. We also had user and kernel spaces.

1. Kernel Level threads
2. User level Threads.

7.1 Differences between thread and processes

1. Process is program in execution, thread is process in execution
2. Inter process communication is slower than inter thread communication.
3. They both have unique ids, unique pid, and unique thread id.
4. Context switching is expensive in processes, it is inexpensive in thread.

5. Every process has its own memory address, but threads use the memory of the process that they belong to.

7.2 User Level Threads

1. user level threads are implemented by users and the kernel is not aware of this existence of these threads.
2. kernel handles them as if they were single threaded processes.
3. User level threads are much faster than kernel level threads.
4. They are represented by a program counter, stack, registers and a small process control block.
5. Also there is no kernel involvement in synchronization for user level threads.
6. User level threads are managed entirely by the user level library.

Advantages of user level threads

1. User level threads are easier and faster to create than kernel level threads, so they can be more easily managed.
2. User level threads can be run on any operating system.
3. There are no kernel mode privileges required for thread switching in user-level threads.

Disadvantages of user level threads

1. Multithreaded applications in user level threads cannot use multiprocessing to their advantage.
2. Entire process is blocked if one user level thread performs blocking operations.

Kernel level threads

1. Kernel level threads are handled by the operating system directly and the thread management is done by the kernel.
2. context information for the process as well as the process threads is all managed by the kernel.
3. Because of this kernel level threads are slower than user level threads.

8 Scheduling

1. Maximum CPU utilization obtained with multiprogramming.
2. CPU IO burst cycle - Process execution consists of a cycle of CPU execution and I/O wait.
3. This IO could be memory or something, it could also be the external device, and stuff.

4. The cycle looks something like : load store
add store
read from file
WAIT FOR IO
store increment
index
write to file
WAIT FOR IO
load store
add store
read from file
WAIT FOR IO

These are CPU and IO burst cycles.

8.1 CPU or Process Scheduling

There are 2 types of scheduling algorithms.

1. Non Preemptive: If the cpu allocated to a process, it keeps the CPU until the process releases it by terminating or switching to waiting state.
 2. Pre Emptive: If the CPU allocated to a process, it may be released if high priority process needs the CPU.
- It may take place in different parts
 - The assigning of the task, or copying its states or something will be non preemptive, so you cant mess with the CPU that time.
 - The actual task execution may be preemptive.

8.2 Scheduling Algorithms

- They deal with the problem of deciding which of the process in ready queue is to be allocated the CPU. Algo compared based on following criteria.
- CPU utilization - Keeping the CPU as busy as possible
- Throughput - Number of processes completed or amount of work done per unit time. This has to be MAX
- Turnaround time - Time of submission of a process at the time of completion. This has to be MIN
- Waiting time - Amount of time a process has been waiting in the ready queue.
- Response time - Amount of time it takes from when a request was submitted until the first response is produced.

8.3 Types of schedulers

Scheduling is broken down into three categories:

1. Long Term Scheduling

- Is performed when a new process is created.
- Long term scheduler or job scheduler selects which processes should be brought into the ready queue.
- Long term scheduler is invoked very infrequently. in seconds or minutes. It may be slow too.
- It controls the degree of multiprogramming.

2. Medium Term Scheduling:

- Part of the swapping function
- Swapping in decisions are taken by medium term scheduler.
- Based on the need to manage the degree of multiprogramming.

3. Short term scheduling

4. Determines which ready processes will be assigned the cpu when it next becomes available and actually assigns the cpu to this process.
5. Short term scheduler or cpu scheduler selects which process should be executed next and allocates cpu
6. short term scheduler is invoked very frequently.

9 Scheduling Algorithms

9.1 First Come First Serve FSFC

Note that this is non Preemptive, so it doesn't care as long as a process is in execution.

1. Selection function, $\max(w)$: selects the process which is waiting in the ready queue for maximum time.
2. Decision Mode: Non Preemptive
3. Throughput : Not Emphasized
4. Response Time : It may be high, especially if there is a large variance in the process execution times.
5. Overhead is Minimum : There's not much logic involved.
6. Effect on Processes : Penalizes short processes.
7. Starvation is Not there.

Example: Processes P1, P2, P3 are there, P1 takes 24 Burst times, P2 takes 3, P3 takes 3. Waiting times:

P1 = 0;

P2 = 24;

P3 = 27;

Average Waiting time:

$$\frac{(0 + 24 + 27)}{3} = 17$$

Turnaround time for P1 = 24, P2 = 27, P3 = 30

Average Turnaround Time =

$$\frac{(24 + 27 + 30)}{3} = 27$$

If you include arrival time, then you have to include it in your calculations. Like you have to subtract it from the turnaround time to find the waiting time and stuff.

9.2 Shortest Job First Scheduling

It associates with each process the length of its next cpu burst. Use these lengths to schedule the process with the shortest time.

There are 2 Schemes here.

1. Non Preemptive: Once CPU given to the process it cannot be preempted until completes its CPU burst.
 2. Preemptive - If a new process arrives with CPU burst length less than remaining time of current execution process, Preempt it. This scheme is known as the Shortest Remaining Time First (SRTF).
 3. It is pretty optimal when it comes to saving the wait times.
1. Selection Function: Minimum total service time required by the process, minus time spent in execution so far.
 2. Decision mode: Preemptive (At Arrival time)