MIT World Peace University

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

Operating Systems

Notes from Tananbaum and Classes

Prepared By

P34. Krishnaraj Thadesar

Batch A2

August 26, 2022

Contents

1 Processes

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

1. Process is compromised of Program Code
2. Data
3. Stack
4. A number of attribute 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 man processes could be waiting for IO
- Swap this proces 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 isnt 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 doesnt take any arguements
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 distinguist 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 wasnt 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 se 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 help 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 threas have separate stacks and individual Thread IDs.
- Thread is a lightweight process because: The contexxt switching between threads is inexpensive in terms of memory and resources.
- Even if 2 processes are communicating within each other, its 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 addreses, priorities.

There are 2 types of threads. This is similar to how there is always this binary existance 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, its 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 existance 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 elvel 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 elvel 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 priviledges 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 suer level thread persforms blocking operations.

Kernel level threads

1. Kernel level threads are handled by the operating system directly and thethread management is done by the kernel
2. context information for the proces 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.