Linnéuniversitetet
Kalmar Växjö

Report

Operating System Concepts
Processes

Semester: Autumn 2019
Course code: 1DV512
Authors: Adam Benn, Nitin Pasikanti, Rashed Qazizada, Adam Rashdan, Munish Sharma and Waseem Totonji
Supervisor: Sabri Pllana

Abstract

Early computers allowed only one program to be executed at a time. This program had complete control of the system and had access to all the system's resources. In contrast, contemporary computer systems allow multiple programs to be loaded into memory and executed concurrently. With respect to these evolutionary changes it resulted in the notion of process which is simply a program in execution. A process is a unit of work in modern computing system. A point to be noted when it comes to processes is that, they require the system resources to be completed like the CPU time, memory, files and in some cases I/O devices. A system therefore consists of a collection of processes, some executing user code, others executing operating system code. This report covers the components of a process and illustrates how they are scheduled in an operating system. It describes how processes are created and terminated in an OS, it also describes and contrasts inter-process communication using shared memory.

Keywords

Process, Process Control Block, Process Scheduling, CPU scheduling, Context Switch, Interprocess Communication, POSIX Shared Memory, Pipes, Windows

Contents

1 Introduction ______ **I**

2 Process Concept ______ **I**
2.1 The Process I
2.2 Process State II
2.3 Process Control Block (PCB) II
2.4 Threads II

3 Process Scheduling ______ **III**
3.1 Scheduling Queues III
3.2 Schedulers III
3.3 Context Switch IV

4 Operations on Processes ______ **IV**
4.1 Process Creation IV
4.2 Process Termination IV

5 Interprocess Communication ______ **V**
5.1 Shared-Memory Systems VI
5.2 Message-Passing System VI
5.2.1 5.2.1 Naming VI
5.2.2 Synchronization VII
5.2.3 Buffering VII

6 Examples of IPC Systems ______ **VIII**

7 Communication between internal processes: ______ **VIII**

8 Communication in client–server systems: ______ **IX**
8.1 Sockets IX
8.2 Pipe mechanism: IX

9 Summary ______ **X**

1 Introduction

The main responsibility of an operating system is the execution of the user program. So, if the operating system is very complex it is expected to do more on the behalf of its users. Hence, a system contains a collection of processes such as system code executed by operating processes whereas user code is executed by user processes. These processes are executing concurrently with one or more CPUs so the management of the system resources is very important. This done by the switching the CPU between the processes, creation and deletion of both user and system process, scheduling of processes and provision of mechanisms for synchronization, communication and deadlock handling for processes. The next section describes process concept and how processes work?

2 Process Concept

Early computers were batch systems that executed jobs, followed by the emergence of time-shared systems that executed jobs, or even tasks. Most modern systems now support multiple threads and as discussed earlier, the management of operating system activities plays a crucial role.

2.1 The Process

As mentioned earlier, a process is a program in execution. The process is not only a program code but also includes the current activity which is represented by the program counter's values and the processor's register's contents. The memory layout of the process is typically divided into multiple sections. These sections are as follows:

- Text section – the executable code
- Data section – global variables
- Heap section – memory that is dynamically allocated during program run time
- Stack section – temporary data storage when invoking functions

A point to note is that the sizes of the text and data sections are fixed, as their sizes do not change during program run time. However, the stack and heap sections can shrink and grow dynamically during program execution. Although the stack and heap sections grow toward one another, the operating system must ensure they do not overlap one another. Another point to note is that a program itself is not a process. A program is a passive entity, like an executable file containing instructions which is stored on disk. A process on the other hand is an active entity with a program counter specifying the next instruction to carry out and a set of associated resources. A program becomes a process when an executable file is loaded into memory. An executable file can be loaded by entering its name on command line or by double clicking an icon representing the executable files.

A program becomes a process when it is loaded into the CPU memory these processes executes in a sequential fashion. Suppose if there are many copies of the same program then each is a separate process as the data, heap and stack sections vary.

2.2 Process State

As a process executes, it changes ‘state’. The state of a process is defined in part by the current activity of that process. The process may be in one of the following states:

- NEW when the process is being created
- RUNNING when instructions are being executed
- WAITING when the process is waiting for some event to occur (such as an I/O completion or reception of signal)
- READY when the process is waiting to be assigned to a processor.
- TERMINATED when process has finished execution.

2.3 Process Control Block (PCB)

Each process is represented in the operating system by a process control block (PCB) also known as a task control block. The following pieces of information is associated with a specific process.

- The state of a process (new, ready, running, waiting etc)
- Program counter which indicates the address of next instruction to be executed.
- The CPU registers which include accumulators, index registers, stack pointers, and general- purpose registers. When an interrupt occurs, the state information must be saved to allow the process to be continued correctly afterward.
- Information related to the process priority and pointers addressing the scheduling queues.
- Memory-management information may include items such as the value of the base and limit registers and the page tables.
- Accounting information includes the amount of CPU and real time used, time limits and so on
- I/O status information includes the list of I/O devices allocated to the process, a list of open files, and so on.

In brief, the PCB simply serves as the repository for all the data needed to start,
or restart, a process.

2.4 Threads

In the above context, so far, we have dealt with a process model that performs a single thread of execution. This single thread of control allows the process to perform only one task at a time. Most modern systems now support multiple threads and thus perform more than one task at a time. This feature is especially beneficial on multicore systems, where multiple threads can run in parallel. The PCB is expanded to include information for each thread on systems that support threads.

3 Process Scheduling

The objective of multiprogramming is always to have some process running to maximize CPU utilization. The time sharing on the other hand lets the CPU core switch among processes quite frequently. Hence, by using process scheduling the system ensures that processes execute efficiently and have reduced wait times.

3.1 Scheduling Queues

The types of queues are discussed below:

- As processes enter the system, they are put into a **job queue**.
- Processes that are in the main memory and waiting to be executed are in **ready queue.**
- Processes which make for an I/O request for a device and needs to wait until the request is handled is stored in **device queue**.

A ready queue header contains pointers to the first and last PCB (Process Control Block) in the list, each of which has a pointer to the next PCB in the list. When a process is selected from the ready queue it will be allocated to a CPU and begin executing. After that it could perform any of the following activities:
If it issues I/O request, it is put in the I/O queue and is in the waiting state. If it creates a new child or a sub-process then it waits for its termination. Finally, it could be interrupted and then goes back to the ready queue. The later mentioned activities are also put back to the ready queue once they are carried out.

3.2 Schedulers

A process migrates between the various scheduling queues throughout its lifetime. The task of the scheduler is to select processes from these queues to execute.

- The role of the Short-term scheduler/ CPU scheduler is to select from among the processes that are in the ready queue and allocate a CPU core to one of them.
- The role of Long-term scheduler/ job scheduler is to select processes from mass-storage device or the Batch system (where data or programs of which have similar jobs are collected) and loads them into the memory for execution.
- Another point to note here is that the CPU scheduler must select a new process for the CPU frequently.
- The main difference between these two schedulers is the frequency of execution (short-term = more frequent)
- Degree of multiprogramming (= the number of processes in memory) is controlled by the long-term scheduler
- Medium-term scheduler removes processes from memory and thus reduces the degree of multiprogramming, later the process can be reintroduced and its execution can be continued from where it left off. This process is can be termed as ‘Swapping’.
- Swapping may be necessary to improve the process mix, or because memory has been overcommitted and needs to be freed up.

3.3 Context Switch

The process of saving the state of old process and switching the CPU to another process is known as context switch. The context of a process is represented in the PCB of a process and this includes the value of the CPU registers, process state and memory-management information. Context switch time is pure overhead as the system does no useful work while switching. The context switch time is highly dependent on hardware support but typically it is a few milliseconds.

4 Operations on Processes

The processes in most systems can execute concurrently, and they may be created and deleted dynamically. The mechanisms for process creation and termination is illustrated below.

4.1 Process Creation

During the course of execution, a process may create several new processes. The creating process is called a parent process, and the new processes are called the children of that process. Each of these new processes may in turn create other processes, forming a tree of processes. The child process can also obtain its resources directly from the Operating system. When a main process created a child process, its execution can be in two ways:

- Parent process continues to execute concurrently with its children
- Parent process waits until child process finishes their tasks.

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

- The child process is the duplicate of the parent process
- Child process has a new program loaded into it.

4.2 Process Termination

A process terminates when it finishes executing its final statement and asks the system to delete it by using exit() system call. At that point, the process may return a status value (typically an integer) to its waiting 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 and reclaimed by the operating system.

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

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

A process that has terminated but whose parent has not obtained the exit status of a child is known as a zombie process. Once the parent obtains the exit status of the child, the process identifier of the zombie process and its entry in the process table are released.

5 Interprocess Communication

Interprocess Communication is the way for processes in an operating system to run at the same time or individually to each other, there are two ways these processes run;

▪ Independently

▪ Cooperating

Independent processes are aptly named because they are independent. This means other processes can run without the interruption of these processes and other processes cannot affect the independent ones. Data between processes that are not shared with others are classed as independent. Cooperating processes are the opposite to independent. Shared data between processes rely on each other so they need to work together. Here are some examples as to why cooperating processes must exist in an operating system;

- **Information sharing** - multiple users may require the use of the same data, a file is an example of such data. An environment must exist so that the multiple user can access the file at the same time without hindrance.
- **Computation speedup** - relies on the fact that a computer has multiple processing cores otherwise speedup is not possible. Computation speedup allows operations to run faster, the operations are divided into smaller parts and are executed adjacent to each other.
- **Modularity** - is building the system in different layers. This separates the processes into different threads of execution.
- **Convenience** - This is so the user has a more user-friendly experience and able to access multiple operations at once.

Processes that operate at the same time need an IPC (interprocess communication) devise, this makes it possible to parse data and information. There are two common ways to do this, firstly, by using shared memory and secondly, using message passing. A place in the memory is allocated with shared memory so that processes can communicate with each other at that location.

To communicate without the use of shared memory space, message passing can be used to communicate between processes instead. Both models have positive and negatives, picking which is used is dependent on things such as; the size of the data being exchanged and the ease of the implementation from a time aspect regarding kernel intervention. Although these methods differ from each other most operating systems include both models.

5.1 Shared-Memory Systems

On completion of memory space allocation for communicating processes, the allocated memory is then accessed in a routine way, this excludes the reliance on the kernel. Shared memory can have performance issues, this could mean less reliance on this model in the future. Most often the location of the shared memory space resides in the address of the process that initiates the communication request. This process goes against the operation systems desired behaviour surrounding memory sharing, this process of sharing memory needs to be agreed upon by the two processes before derestricting the operating system's sharing preferences. Once this process is complete the processes can begin to share data in the allocated space.

The communication that occurs in this space is reading and writing of data, the operating system is not in charge of the data shared as that responsibility now resides with the two communicating processes. To access the location of shared memory the need of designed code specifically for this task needs to be created. The use of buffers such as unbounded and bounded buffers are used to let processes run at the same time, these buffers will be in the location of the shared memory used by the two processes.

5.2 Message-Passing System

Message Passing System allows communication between processes in a message passing environment. This process does not require the sharing of a memory location and it is a good way of communicating over a network. It is just like sending any type of "message" there is a sender and a receiver with messages differing in size. This size difference in data can make implementation relatively simple to more difficult, it can be both fixed and vary in size.

A link must be in place so the processes can send and receive messages from each other. The send and receive tasks can be carried out in the following ways;

- Direct or indirect communication
- Synchronous or asynchronous communication
- Automatic or explicit buffering

These processes help to create a communication link between process and the creation of these links can be in the form of physical or logical links.

5.2.1 Naming

Naming is a way of identifying processes that wish to send and receive messages to and from each other. There are two ways to achieve this, by direct or indirect communication. With direct communication the processes involved must name the receiver and the sender of the message. There are certain properties a direct communication connection has. An automatic link is created between all the processes wanting to communicate, each process must be able to identify each other for this to happen. Once a link has been established that link is only meant for the pair of processes that required the connection. When using the direct approach, the processes can use two ways of addressing each other. The first approach is addressing symmetrically and the second is asymmetrically. The only difference between the two is the way the addressing occurs.

Indirect communication is called indirect due to the fact the messages to be sent and received end up in a port, otherwise called a mailbox, which is shared by the processes wishing to pair. Once the mailbox has been created it only exists for the duration it is required for, this happens when the owner process terminates it. Using this method makes it possible to have links with multiple processes. The responsibility of creating a mailbox lies with the operating system, here’s what the operating system should provide;

- Creation of mailboxes
- Send and receive messages through the created mailbox
- Deletion of a mailbox

5.2.2 Synchronization

Synchronization is a way of passing messages. The passing messages can be blocking or non-blocking, these are also known as synchronous and asynchronous passing. Both these passing techniques can be applied to the sending and receiving of messages.

Blocking of the sending process blocks the sender whilst waiting for the message to be received, blocking receive blocks messages until they are accessible to the receiver. Non-blocking send works a little different, it is not as restrictive. The sent message gets passed then the send process continues its own operation. The receive action is not hindered either, if a message is there it can be taken if not null will be returned to the receiver. There are other ways to pass the messages, a process called rendezvous can be used. Rendezvous is when both the participants in message passing are blocking.

5.2.3 Buffering

It does not matter which means of communication is used between processes, direct or indirect, the use of buffers is required. Buffers are queues that temporarily store the messages, there are three different types of queues that could be used, they are;

- Zero capacity – the maximum length is zero, messages cannot be in a waiting state. The use of blocking send must be used. This method offers no buffering due to the length.
- Bounded capacity – the length is fixed so the number of messages that can be stored is the same of the queue length. With a full queue blocking the sender happens until the message has space to enter.
- Unbounded capacity – Does not need to use blocking, it is in principle infinite, so the number of messages that can be stored is the same.

6 Examples of IPC Systems

As explained before, IPC is a virtual communication technique between different processors in different operating environments. We can summarize the examples of internal communication between operations, into the following points: It is understood that processes are programs that the operating system executes, So these processes must be communicated between them, and this type of communication is called an internal communication, i.e. the ability of one process of communication to exchange information and data with another process.

But since operations are just a series of code or programs, one process can communicate in different ways and each process performs a particular task and provides it as a service to other processes it requires. As an example, a web server must have one operation to listen for network requests from browsers and to show HTML pages, it may need another process to serve complex data requests and may also need another process to process ftp file transfer requests and each process is designed to do one task to the fullest.

This allows the network administrator to share the task that an operation provides on a set of processes. For example, if there are too many FTP requests, it distributes these requests between two processes: Shared memory: POSIX Shared Memory: Sharing a virtual memory between two or more processes, so data transfer and communication between all processes are transferred. 2 - Massage-passing messages: It is a message between operations to exchange data, but before the transmission must be determined the sender and the receiver and also must be between the two loops to be exchanged.

7 Communication between internal processes:

Internal communication between processes can be likened to the user and the server. and this communication can be as a form or series of a network of processes, each packet is sending a message to the other "peer - to - peer".

A server creates a named server connection port object. A client makes a connect request to this port. If the request is granted, two new unnamed ports, a client communication port and a server communication port, are now created. The client gets a handle to the client communication port, and the server gets a handle to the server communication port.

The client and the server will then use these new ports for their communication. When a server wants to read or write larger amounts of data than will fit in a shared section, data can be directly read from or written to a client’s address space

An Example: Windows OS

Windows operating system is an example of modern design that employs modularity to increase functionality and decrease the time needed to implement new features. Windows provides support for multiple operating environments, or subsystems. Application programs communicate with these subsystems via a message-passing mechanism as per as the following:

- For small messages (up to 256 bytes), the port's message queue is used as intermediate storage, and the messages are copied from one process to the other
- Larger messages must be passed through a section object, which is a region of shared memory associated with the channel.
- When the amount of data is too large to fit into a section object, an API (Application performance interface) is available that allows server processes to read and write directly into the address space of a client.

8 Communication in client–server systems:

8.1 Sockets

Communication using sockets—although common and efficient—is low-level form of communication between distributed processes. One reason is that sockets allow only an unstructured stream of bytes to be exchanged between the communicating threads. It is the responsibility of the client or server application to impose a structure on the data.

8.2 Pipe mechanism:

It can be likened to water channels in the house, where there is a stream in which dataflows from one side and flows from the other. The channel is somewhat similar to the file in terms of dealing with the data as a flow of successive data, but different from the file that it has two ends, when creating a channel of communication, we get two points to the end. The other difference does not have a physical store of data as in the file, when you close the channel, any information has been written at one end did not read the second end will be lost. The Pipes strategy can be illustrated as an "information stream" between the two processes.

Here we see two processes, "Parent and Child". Each channel can be used to send requests or retrieve information depending on which process first started contacting.

A pipes strategy typically provides one of the simpler ways for processes to communicate with one another, although they also have some limitations. In implementing a pipe, four issues must be considered:

- ➢ Does the pipe allow bidirectional communication, or is communication unidirectional?
- ➢ If two-way communication is allowed, is it half duplex (data can travel only one-way at a time) or full duplex (data can travel in both directions at the same time)?

- Must a relationship (such as parent–child) exist between the communicating processes?
- Can the pipes communicate over a network, or must the communicating processes reside on the same machine?

9 Summary

A process is a program in execution. As a process executes, it changes state. The state of a process is defined by that process's current activity. Each process may be in one of the following states: new, ready, running, waiting, or terminated.
Each process is represented in the operating system by own process control block (PCB).

A process, when it is not executing, is placed in some waiting queue. There are two major classes of queues in an operating system: I/O request queues and the ready queue. The ready queue contains all the processes that are ready to execute and are waiting for the CPU. Each process is represented by a PCB. The operating system must select processes from various scheduling queues. Long-term (job) scheduling is the selection of processes that will be allowed to contend for the CPU. Normally, long-term scheduling is heavily influenced by resource-allocation considerations, especially memory management. Short-term (CPU) scheduling is the selection of one process from the ready queue.

Operating systems must provide a mechanism for parent processes to create new child processes. The parent may wait for its children to terminate before proceeding, or the parent and children may execute concurrently. There are several reasons for allowing concurrent execution: information sharing, computation speedup, modularity, and convenience.
The processes executing in the operating system may be either independent processes or cooperating processes.

Cooperating processes require an interprocess communication mechanism to communicate with each other. Principally, communication is achieved through two schemes: shared memory and message passing. The shared-memory method requires communicating processes.

Bibliography

[1] Abhraham S, Peter B. G, Greg G, Operating System Concepts, 10th ed, Wiley, 2018, pp. 105- 156.

Team Work

The entire was active from the beginning till the end of the Theoretical assignment. We communicated through Slack and divided tasks amongst our team members based on our strengths and weaknesses. Through effective communication and planning of our tasks we could finish the report as well as the presentation slides on time.