AHMEDABAD UNIVERSITY
Global Education at Local Cost, Context and Ethos™

OPERATING SYSTEMS LAB PROJECT REPORT

Title: "Implementation of Process Control Block"

Guide: Prof. Sanjay Chaudhary

Team members:

Akash Shah - 1401047

Deep Parekh - 1401001

Kashish Shah - 1401048

Raj Shah - 1401050

AHMEDABAD UNIVERSITY | School of Engineering and Applied Science

AHMEDABAD UNIVERSITY
Global Education at Local Cost, Context and Ethos™

TABLE OF CONTENTS

BRIEF DESCRIPTION

1) What is a Process?

• It is defined as a program in execution and can also be known as an instance of a program running on a personal computer.
• It can also be defined as the entity that can be executed or assigned to a mere processor.
• A process is a unit of activity which can be characterized by its current state, set of system resources and execution of a sequence of instructions.

At any given point in time, *while the program is executing,* this process can be uniquely characterized by a number of elements, including the following:

• **Identifier:** It is a unique kind of identifier which is associated with the process so that it can distinguish it from all other processes that arrive.
• **State:** A process is said to be in a running state, if it is currently executing.

• **Priority:** it is the priority level of current process relative to the priority of other processes.

• **Program counter:** The program counter is the address of the next instruction in the program to be executed.

• **Memory pointers:** These include pointers to the program code and data associated with this process.

• **Context data:** this refers to the data that are present in registers of the processor while the process is under execution.

• **I/O status information:** This includes outstanding I/O requests, I/O devices, assigned to this process and the list of files which are in use by the process.

• **Accounting information:** This part of the process control block includes the clock time used, time limits, account numbers and the amount of processor time.

The information in the preceding list is stored in a data structure, typically called a **PROCESS CONTROL BLOCK**

1) PROCESS CONTROL BLOCK

It is a data structure in the operating system kernel containing the information which is needed to manage different processes. It stores many different items of data necessary for a proper and efficient process management.

1. Process state
2. Process id
3. Process privileges
4. Program Counter
5. Stack pointer
6. CPU registers
7. Memory management information
8. Accounting information
9. I/O status information

- PCB is created and managed by the OS.
- It contains sufficient information so that it is possible to interrupt a running process and later resume execution as if the interruption had not occurred.
- It is the key tool that enables the OS to support multiple processes and to provide for multiprocessing.
- During interruption of a process, the current values of the processor registers and program counter are saved in fields of the process control block, and the state of the process is changed to the latest state.

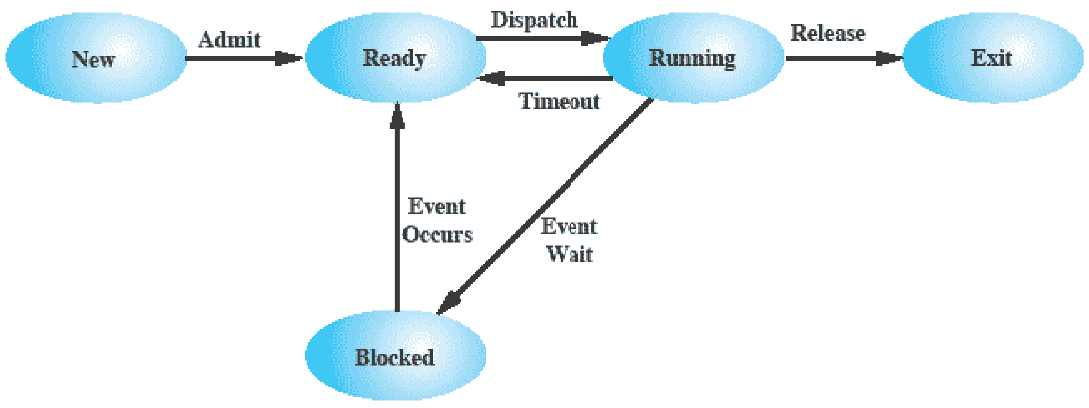


Fig 1: Simple Process state model

• **Running:** The process that is being executed.
• **Ready:** A process that is prepared to get executed when the opportunity arrives.
• **Blocked/Waiting:** A process in this state cannot execute until some event occurs.
• **New:** A process that has just been created but not yet been categorised under the group of executable processes.
• **Exit:** A process that has been released from the category of executable processes by the OS, because it halted or has been aborted.

BLOCK DIAGRAM

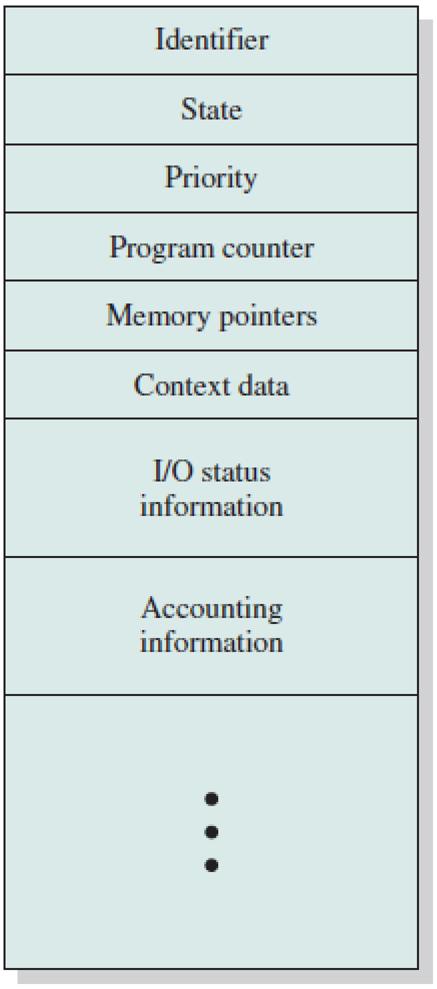


Fig 2: Simplified Process Control Block Diagram (PCB)

The architecture of a PCB is completely dependent on Operating System and may also contain some different information in different operating systems. The PCB is maintained for a process throughout its lifetime, and it is deleted when the process terminates.

ALGORITHMS AND CONCEPTS USED

Here, we have mentioned all the concepts and algorithms utilized in the program along with their implementation.

Structures: They are basically utilised for the declaration of complex data types to group a list of variables to be placed under one name in a memory block. It allows different variables to be accessed via a single pointer. This data structure has been used to form the structure of the process control block and its associated elements (ID, registers etc..).

Threads: A thread is basically a process within a process. A process can be divided into multiple thread processes to carry out different functionalities associated with the process. A thread has a program counter which keeps a track of the scheduling of the execution of different instructions, it has a stack which contains the history of the instructions which have been executed and registers to hold the working variables.

All threads have code segment, data segment and open files in common. When one thread makes amendments, that change is reflected in all other threads within the same process.

Random number generator: This is used to assign the process id randomly to a particular process. This number is generated by an algorithm that returns a sequence of apparently non-related numbers each time it is called. The function is rand () which is an inbuilt function in c.

Interrupt signal: When a signal is sent, the operating system interrupts the normal flow of execution of the current process to deliver the signal. If the signal has an associated signal handler, it gets executed or else there is an interrupt.

In this project, **Ctrl + C** is devised as an external interrupt provided by the user, which stops the execution of the current process, dequeues the current process and enqueues it at the end of the queue and it takes the process in the top of the queue for execution.

Time slicing: time slicing is the time period for which a process has been allowed to run without any interruption before the pre-emption takes place. Once for every time slice scheduling is done by the scheduler to decide which would be the next process to run. The time slice should be decided upon very precisely as if it is too short, too much processing time will be consumed by the scheduler but if it is too long then the response to external events may not be quick enough. In our project time slicing, has been implemented in a loop manner where in a particular loop iterates for a particular number of times which is the permitted time for the process execution.

File Handling: File management system consists of system utility programs that run as privileged applications. It is Concerned with secondary storage. File handling has been implemented in reading of a test text file case which is the test process itself. Also, it is used in the generation of log files which displays the process id and the process states.

Forking: Forking is done when a process wants to create a copy of itself. Forking is categorised as a system level call which is done by the kernel. Here we have just demonstrated how the process of forking takes place and how the pids' and ppids' are assigned.

Race Condition: Race condition is the condition where in there is a multiple reading or writing of different data items by more than one processes or threads. The output at the end depends on the execution time of each operation performed and which particular thread or process has finished last. Here, we have used six processes to demonstrate the principles of race condition. Two processes of addition, two of subtraction and two for multiplication. These processes are selected on the basis of their priority and the output depends on who finishes the race last.

Mutual Exclusion: mutual exclusion is concept implemented when we have concurrency issues. It is a process of restricting one process from accessing a shared resource while another process is utilizing that resource. Only after one resource has finished using the resource; another process can be allowed to perform modifications on it. So, this excludes different processes to execute or modify the shared data or variables simultaneously. Four conditions are necessary to implement mutual exclusion:

- No two processes should be in the critical sections at the same time.
- The relative speeds of processors and the number of CPUs have not been taken under considerations or assumptions.
- No process outside the critical section should interfere in the execution of other processes.
- No process should be kept waiting the critical section if there is no other process accessing it.

In our project, mutual exclusion has been implemented by the creation of threads. Two threads have been created for performing addition. If mutex_lock is true mutual exclusion is exercised and the execution of second thread takes place only after the execution of the first thread and if the lock is not present execution and processing takes place simultaneously.

Scheduling Algorithms: The job of a process scheduler is to schedule different process to be assigned to the central processing unit. This is done on the basis of different scheduling algorithms.

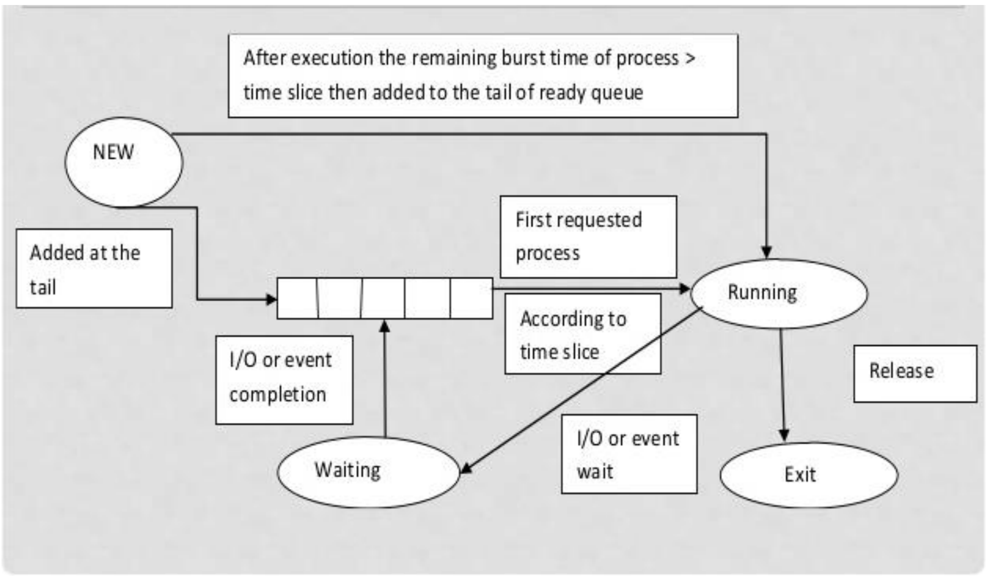
There are two types scheduling algorithms. Pre-emptive and non-pre-emptive scheduling algorithms.

Non-pre-emptive algorithms are designed such that If a process enters a running state it cannot be pre-empted before its allotted time completes.

Pre-emptive algorithms are designed based on the priority. A running process can be pre-empted if it has a low priority and a high priority process has entered the ready state.

- **Round robin scheduling**

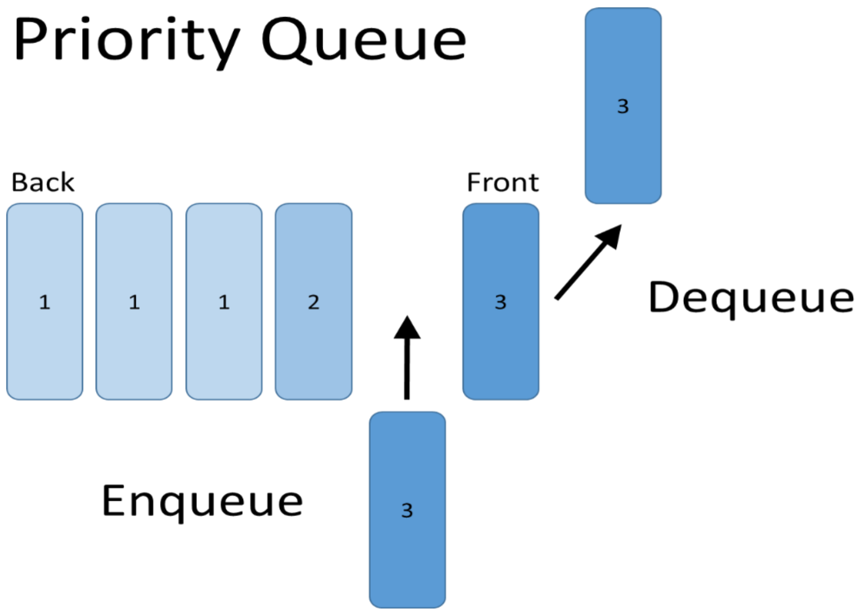
Flow chart:



As the term is generally used, time slices are assigned to each process in equal portions and in circular order, handling all processes without priority. It is a pre-emptive process scheduling algorithm. Each process has been provided with a fixed slot of time to execute, known as a **quantum**. A process gets pre-empted once it has been executed for the given time period. To save states of pre-empted process, context switching is used. In our project, there are three processes into consideration for which reading and execution is done in a round robin fashion.

- **Priority Queue based scheduling**

It is a data structure which has a sequence of objects/data that are waiting to be processed. Here we have used a queue to enqueue and dequeue processes in the order of their priority in the process control block. An element with high priority is processed before an element with low priority. If two elements have the same priority, then they are considered according to order in which they appear in the queue. Once a process gets dequeued from the top, its priority decreases by 1. We have considered six processes to demonstrate priority queue scheduling in our project.



FLOW OF THE CODE

Here the flow in which the project was built with addition of certain elements has been explained in a sequential manner.

- Firstly, the structure of process control block was created specifying the various components and their functions essential for the functioning of the process control block on a c platform using the structures data type.
- Then a test file was taken into account which basically had numbers in a sequential manner from 1 to 10 lakh.
- So now to deal with the test file, concepts of file handling were introduced for reading the test file and also for the generation of log files to store the process id and process states.
- In our project, we have taken three processes P1, P2 and P3 into account to demonstrate the functioning of the process control block.
- The file read functions associated with the processes are performed in a round robin fashion according to the round robin algorithm. For this algorithm to function, queue data structure was added which was responsible for the enqueuing and dequeuing of the processes.
- The output will be the count of the program counter of the process which is stored in the log file mentioning the process id and the process state of the process which is being dequeued.
- The read functions in the round robin manner are made to run continuously in a loop manner until it reaches the end of the file.

- The concept of signal interrupt was then introduced which was performed while the read functions read from the test file. In this project, **Ctrl + C** is programmed as an external interrupt provided by the user, which stops the execution of the current process, blocks the current process, dequeues the current process and enqueues it at the end of the queue and it takes the process which is on the top of the queue for execution.
- Another concept implemented next was the priority queue. In this case processes are added into a priority where the process to be executed depends on the priority assigned to it. A process here terminates depending upon the time slice assigned to it and the next process is chosen depending upon its priority. Here also a different log file is maintained which stores the process id and state of the processes.
- Next the concept of race condition was taken into account. For showcasing its implementation six processes in a priority queue were taken into consideration. Two processes perform addition, two processes perform subtraction and two processes perform multiplication. The process that has to be chosen for execution out of these six processes is dependent on the priority of that process. Here the final result depends on the order of execution of the processes and the output depends on who finishes the race last.
- Lastly the concept of mutual exclusion was implemented through the creation of threads. Two threads have been created in order to show mutual exclusion. Both perform addition operations. Mutual exclusion is exercised through a condition named mutex_lock. If the condition is valid first thread will complete its execution and give the output after which second thread will complete its execution and give the

output. In this case second thread would not intervene when the first thread is under execution. If the condition is not valid then the processing of the threads will be done simultaneously and one thread can interfere the processing of another thread.

- Finally, then .h files are created for the files for data abstraction purposes. So, .h files are created for file read functions and the declaration of the structure of process control block to hide the background details from the user.

TEST DATA SETS

- In our project for the test case we have taken a text file **temp.txt** which contains numbers in a sequential manner listed from 1 to 10 lakhs. The numbers would be read from this file in a proper sequential manner and upon this process other concepts of operating systems would be implemented.

IMPLEMENTATION AND RESULTS

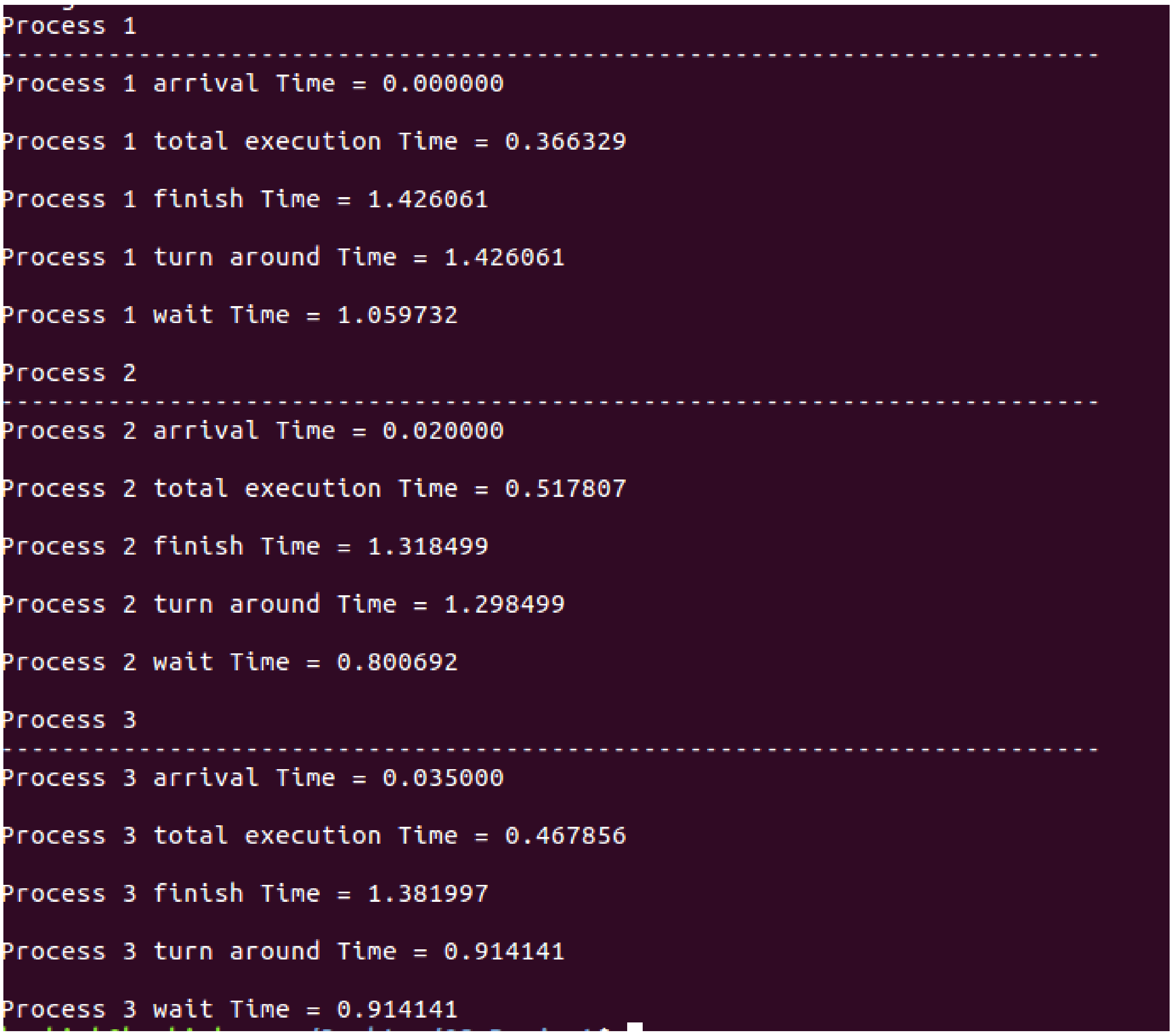
The following files have been incorporated in the project; each filename has a description for the contents and the result it would produce:

- **PCB_structure.h** - it contains the declaration of the basic structure of the process control block along with its components and instances of the processes.
- **pcb.c** - It contains the implementation of the process control block with respect to three processes. The processes are inserted in a queue. The process on the top of the queue is chosen for execution which reads the contents from the test file and after a given time slice the process is dequeued and is enqueued from the tail of the queue. The next process on the top is chosen next. Thus, it follows round robin scheduling algorithm.
- **pcb_fileread.h**- This contains the functions for reading from the test file for the above mentioned pcb.c
- **pqpcb.c-** It contains the implementation of the process control block with respect to six processes. The processes are inserted in a queue according to their priority assigned. The process in the queue is chosen for execution depending upon its priority order which reads the contents from the test file and after a given time slice the process is dequeued and is enqueued from the tail of the queue. When the process is dequeued from the queue; its priority gets decreased by 1 as it has been processed once.

The next process on the top of the priority list is chosen next. Thus, it follows priority queue scheduling algorithm. (greater the priority number, higher the priority).

- **pqpcb_fileread.h-** This contains the functions for reading from the test file for the above mentioned pqpcb.c
- **rcpcb.c-** This code implements the concept of race condition by taking into account six processes. Two processes perform addition, two processes perform subtraction and two processes perform multiplication. The process that has to be chosen for execution out of these six processes is dependent on the priority of that process. Here the final result depends on the order of execution of the processes and the output depends on who finishes the race last
- **mutex.c-** This code implements mutual exclusion where in two threads are defined using the p_thread function. Two threads have been created to perform addition operations. Mutual exclusion is exercised through a condition named mutex_lock. If the condition is valid first thread will complete its execution and give the output after which second thread will complete its execution and give the output. In this case second thread would not intervene when the first thread is under execution.
- **log.txt-** it contains the output for pcb.c
- **log2.txt-** it contains the output for pqpcb.c
- **log3.txt-** it contains the output for rcpcb.c

OUTPUT ANALYSIS



```
Process 1
--------------------------------------------------------------------------
Process 1 arrival Time = 0.000000

Process 1 total execution Time = 0.366329

Process 1 finish Time = 1.426061

Process 1 turn around Time = 1.426061

Process 1 wait Time = 1.059732

Process 2
--------------------------------------------------------------------------
Process 2 arrival Time = 0.020000

Process 2 total execution Time = 0.517807

Process 2 finish Time = 1.318499

Process 2 turn around Time = 1.298499

Process 2 wait Time = 0.800692

Process 3
--------------------------------------------------------------------------
Process 3 arrival Time = 0.035000

Process 3 total execution Time = 0.467856

Process 3 finish Time = 1.381997

Process 3 turn around Time = 0.914141

Process 3 wait Time = 0.914141
```

This is the output of **pcb.c** where a process has been evaluated on various parameters. The parameters include:

- arrival time - time at which process is submitted
- total execution time - burst time
- finish time - time at which process is completed
- turnaround time - time required by a process from submission to completion
- wait time - time for which process waits in queue waiting for its turn to get executed.

ACKNOWLEDGEMENT

We would like to express our appreciation to all those who provided us their support and valuable time to complete this project. A special gratitude we give to our professor of Operating systems Prof. Sanjay Chaudhary and the concerned teaching assistants whose contribution in stimulating suggestions and encouragement helped us to coordinate our project successfully.

REFERENCES

- https://en.wikipedia.org
- Operating Systems: Internals and Design Principles, 6/E William Stallings
- http://www.dictionary.com
- https://www.tutorialspoint.com/
- http://image.slidesharecdn.com/
- http://www.sanfoundry.com/
- https://users.cs.cf.ac.uk/Dave.Marshall/C/node31.html
- Lecture notes
- www.oracle.com
- http://man7.org/linux/man-pages/man7/pthreads.7.html
- www.geeksforgeeks.org
- http://www.webopedia.com/TERM/M/mutex.html
- https://users.cs.cf.ac.uk/Dave.Marshall/C/node24.html
- https://netmatze.files.wordpress.com/2014/08/priorityqueue.png