

CSE-316

**FOUR LEVEL PRIORITY DISPATCHER**

K18PD-G2-65

Student Name: Lov Kumar

Student ID : 11804637

Email Address: lovkumar33@gmail.com

GitHub Link: <https://github.com/lovkumar33/OS-Assignment>

Submitted To :

Amandeep Kaur

**Q35**. Write a four level priority process dispatcher operating within the constraints of finite available memory and I/O resources. Use different dispatchers and memory allocation.



**Solution : -**

Scheduling is the activity of process manager that handles the removal of the running process from the central processing and choose other process on the basis of a particular strategy.

Scheduling is the important part of a Multiprogramming OS. These operating system allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

**Dispatcher :**

Dispatcher is the other component that is involved in the CPU-scheduling, which is the module that gives control of the CPU to the process selected by the short-term scheduler. This receives control in kernel mode as the result of an interrupt or system call. Functions of a dispatcher mop the following:

1. [Context switches](https://en.wikipedia.org/wiki/Context_switch), in which the dispatcher saves the [state](https://en.wikipedia.org/wiki/State_(computer_science)) (also known as [context](https://en.wikipedia.org/wiki/Context_(computing))) of the [process](https://en.wikipedia.org/wiki/Process_(computing)) or [thread](https://en.wikipedia.org/wiki/Thread_(computing)) that was previously running; the dispatcher then loads the initial or previously saved state of the new process.
2. Switching to user mode.
3. Jumping to the proper location in the user program to restart that program indicated by its new state.

Dispatcher must be as fast as possible, As it is invoked during every process switch. At the time of context switching, the processor is virtually idle for a fraction of time, Hence unnecessary context switches should be avoided. Time consumed for the dispatcher to stop 1 process and start other is dispatch latency.

**Problem Explained :**

Problem of scheduling of resources in manufacturing system has been studied over the last four decades. Research work has been carried out on the theory of scheduling and its practical applications. Methods based on branch and bound technique, priority dispatching rules, simulated annealing, tabu search, genetic algorithm, artificial neural networks, expert systems, linear programming, integer programming, graph theory, petri nets and lagrangian relaxation techniques etc., have been proposed. Objectives such as the minimization of lead time and work in process inventory, and maximisation of throughput, resource utilisation and timely delivery have been considered.

The scheduling studies into the following two

categories :

* Theoretical research dealing with optimisation procedures.
* Experimental research dealing with dispatching rules.

Some theoretical research has focused at developing mathematical model and optimal or

Sub-optimal algorithms. The theoretical result have not been widely used in the industries due to the huge computational complexity of the scheduling problem. The experimental research has been primarily concerned with priority dispatching rules and heuristic that can efficiently solve the scheduling problems.

**Processes :**

Processes on User are simulated by the dispatcher creating a new process for each dispatched process. These process is a generic process that can be used for any priority process. It actually runs itself at very low priority, sleeping for one-second periods and displaying the following:

1. A message displaying the process ID when the process starts;
2. A regular message every second the process is executed; and
3. A message when the process is Suspended, Continued, or Terminated.

These Process will terminate of its own record after 20 seconds if it is not terminated by your dispatcher. The process prints out using a randomly generated color scheme for each unique process, so that individual "slices" of processes can be easily distinguishable. Use this process rather than your own.

The life cycle of a process is as follows:

The process is submitted to the dispatcher input queues via an initial process list that designates the arrival time, priority, processor time required (in seconds), memory block size, and other resources requested.

A process is "ready-to-run" when it has "arrived" and all required resources are available.

Any pending Real-Time jobs are submitted for execution on a first-come-first-served basis.

If large resource and memory are available for a low priority User process, the process is transferred to the appropriate priority queue within the feedback dispatcher unit, and the remaining resource indicators (memory list and i/o devices) updated.

When a job is started (fork and exec("process",...)), the dispatcher will display the job parameters (Process ID, priority, processor time remaining (in seconds), memory location and block size, and resources requested) before performing the exec.

A Real-Time process is allowed to run until its time has expired when the dispatcher kills it by sending a sigint signal to it.

lower priority User job is allowed to run for one dispatcher tick (one second) before it is suspended or terminated if its time has expired. When suspended, their priority level is lowered (if possible) and it is requeued on the appropriate priority queue as shown in Figures. To retain synchronization of output between your dispatcher and the child process, your dispatcher should wait for the process to respond to a SIGTSTP or SIGINT signal before continuing. To match the performance sequence indicated in the comparison of scheduling policies, the User job should not be suspended and moved to a lower priority level unless another process is waiting to be restarted.

Provided no high priority Real Time jobs are pending in the submission queue, the highest priority pending process in feedback queues is started or restarted.

Whenever a process is terminated, the resource it used are returned to the dispatcher for reallocation to next processes.

When there are no more processes in the dispatch list, the input queues and the feedback queues, the dispatcher exits.

**Dispatch List :**

Dispatch List is a list of processes to be processed through the dispatcher. The list is contained in a text file that is specified on the command line. That is,

>hostd dispatchlist

Every line of the list describes one process with the following data as a *"comma-space*" delimited list:

<arrival time>, <priority>, <processor time>, <Mbytes>, <#printers>, <#scanners>, <#modems>, <#CDs>

Thus,

12, 0, 1, 64, 0, 0, 0, 0

12, 1, 2, 128, 1, 0, 0, 1

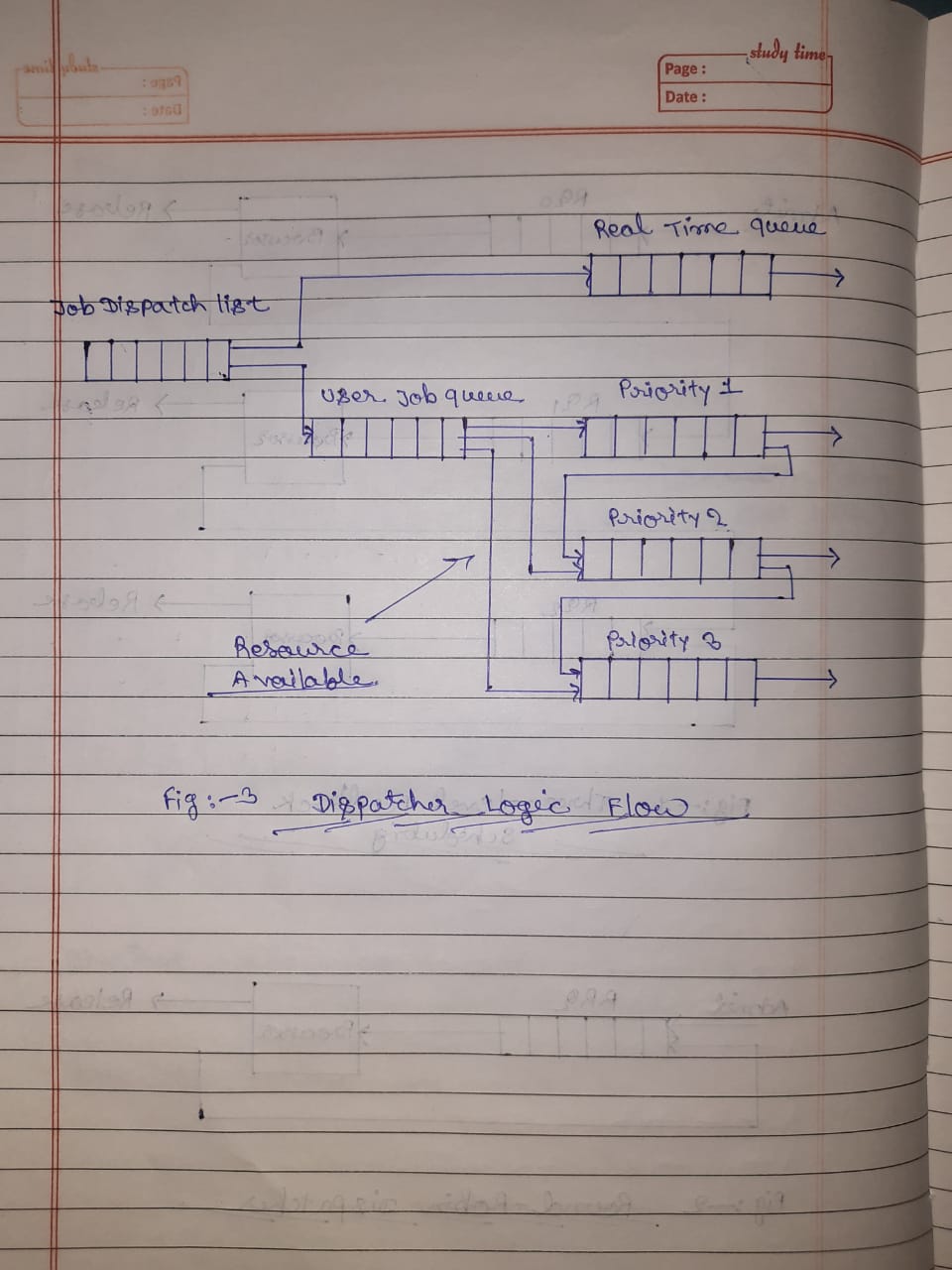
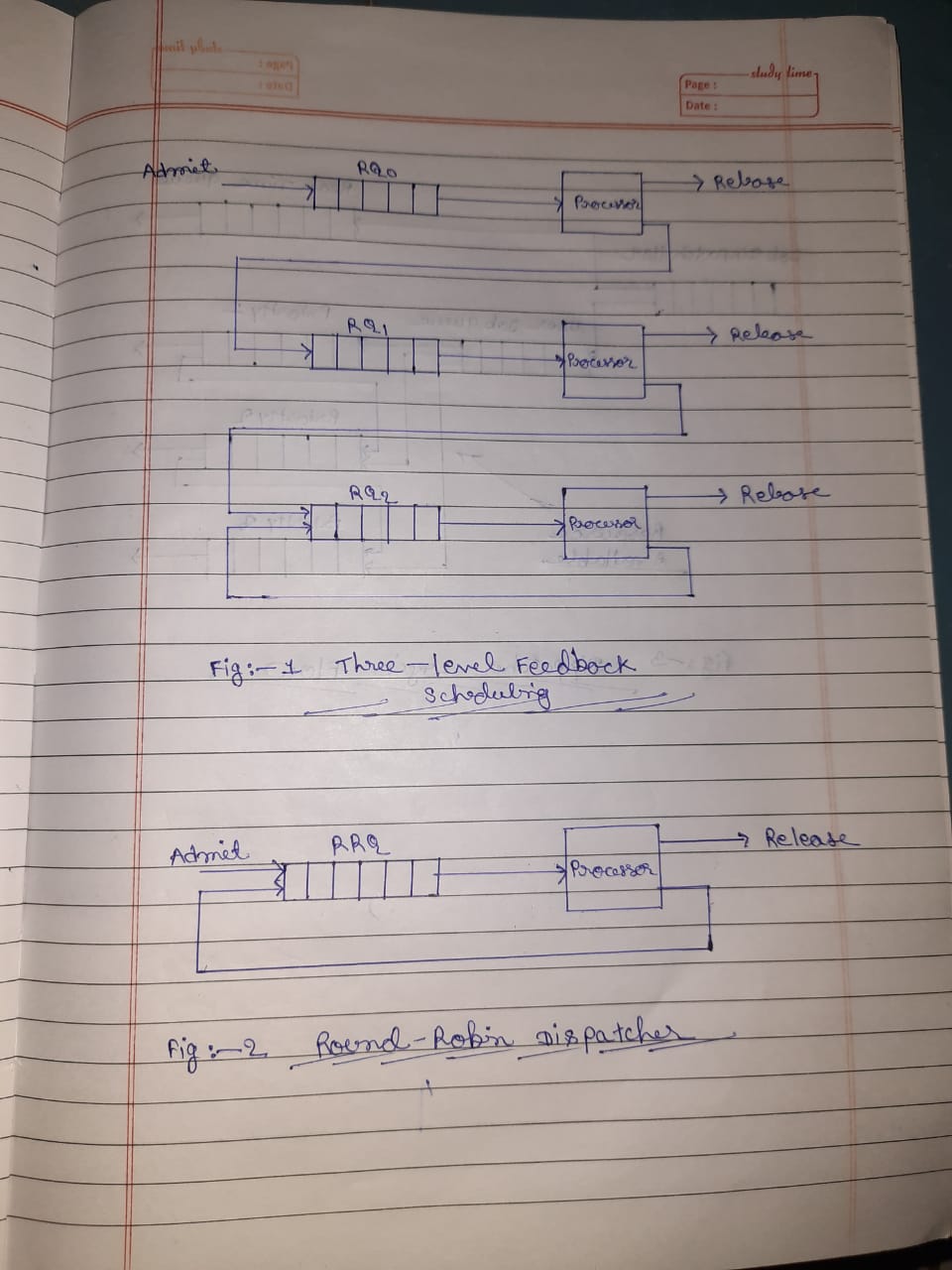
13, 3, 6, 128, 1, 0, 1, 2

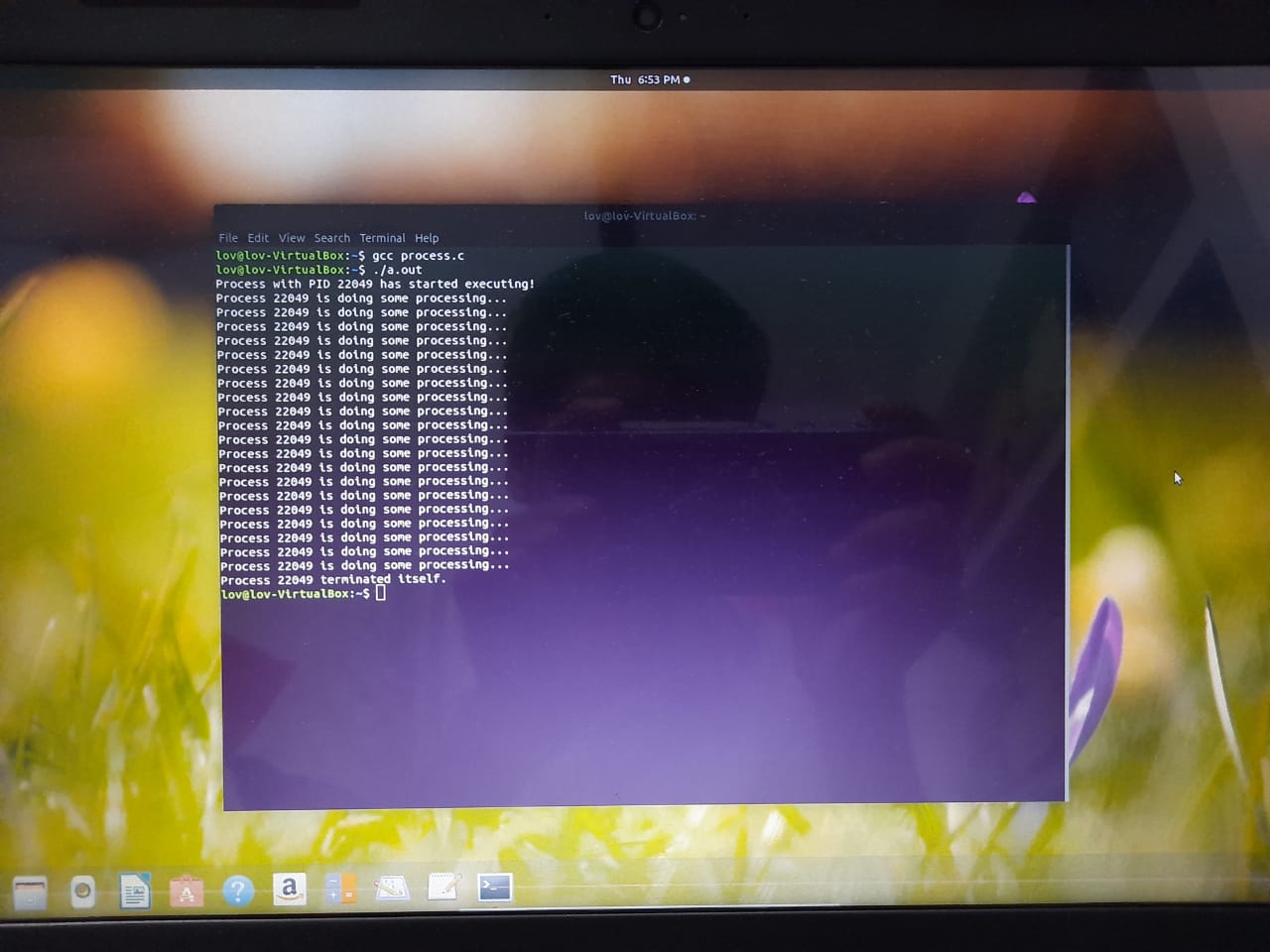
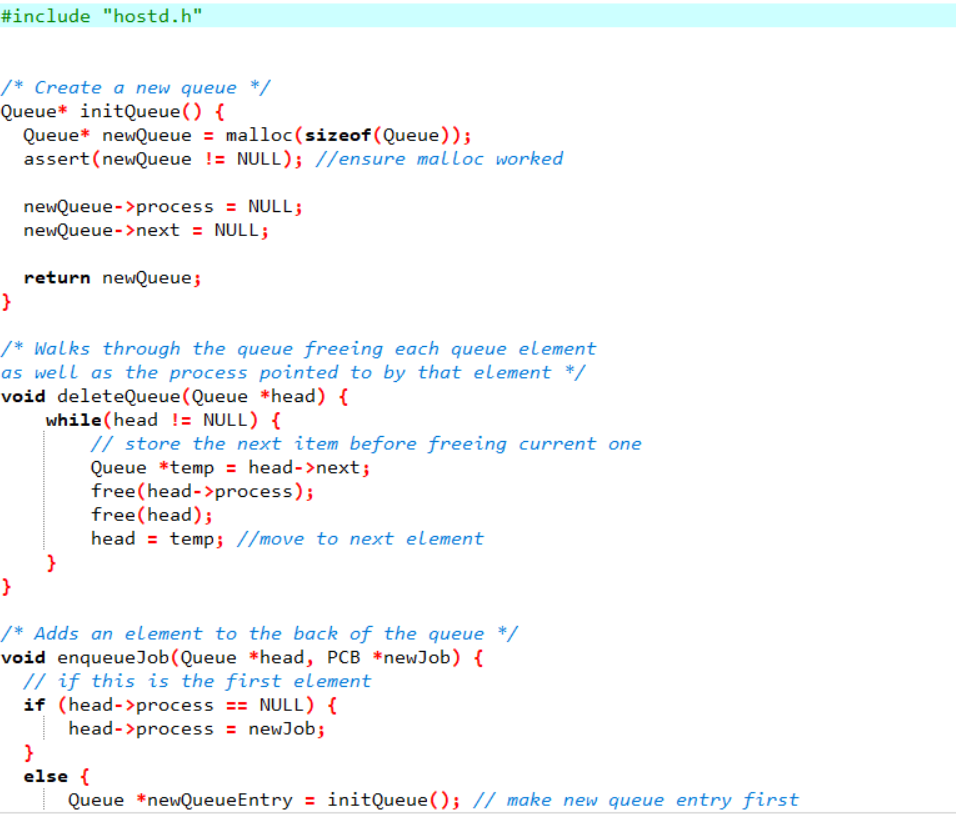
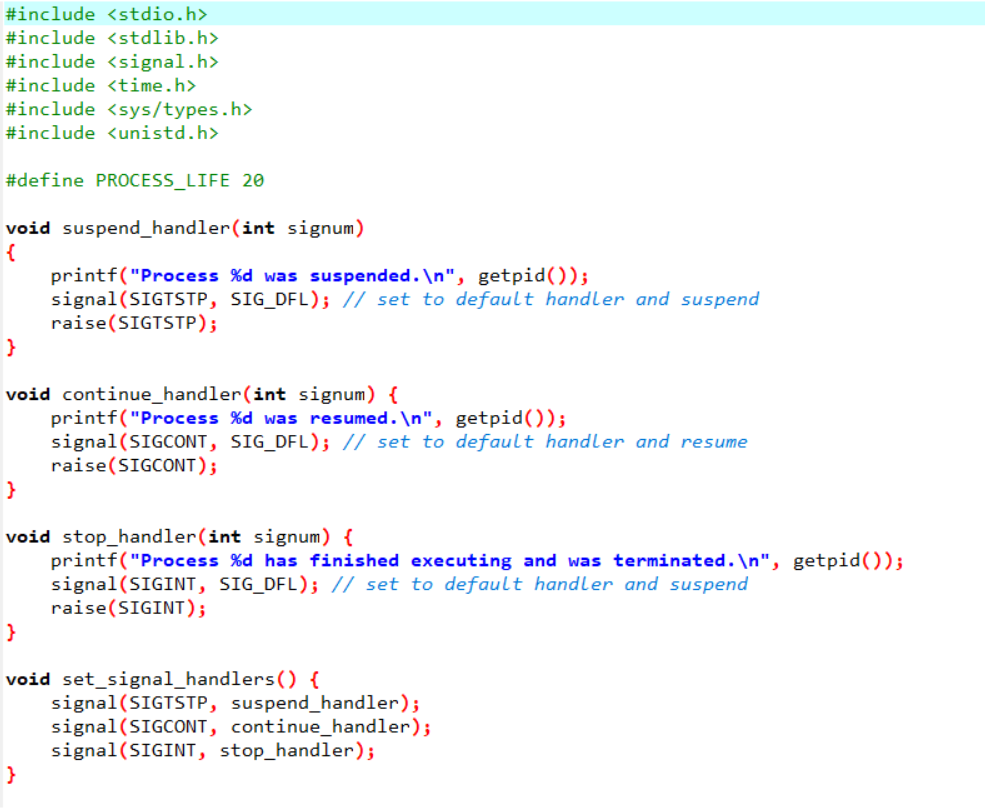
would indicate the following:

**First Job:** Arrived at 12, priority 0 (Real-Time), required 1 second of processor time and 64 Mbytes memory — no I/O resources required.

**Second Job:** Arrived at 12, priority 1 (high priority User job), required2 seconds of processor time, 128 Mbytes of memory, 1 printer, and 1 CD drive.

**Third Job:** Arrived at 13, priority 3 (lowest priority User job), required 6 seconds of processor time, 128 Mbytes of memory, 1 printer, 1 modem, and 2 CD drives.





Code snap

Output