RADHIKA PATWARI

18CS10062

PAGE NO.
DATE 31 | 3 | 2021

# Take Home Assignment 2

1:

```
/* including header files */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>          /* for vfork() syscall */
#include <sys/wait.h>        /* for wait() syscall */
#include <sys/shm.h>
#include <sys/types.h>

/* function to count the number of words in a file pointed
   by file descriptor file */
/* using '\n' and ' ' as indicator of word separators */
int countWords (FILE *file) {
    int count=0, seen=0;
    char character;

    while((character = fgetc(file)) != EOF){

        if(character==' ' || character=='\n' || character=
=='\r' || character == '\t' || character=='.' || character==
'?' || character=='!')   seen=1;
        else if (seen) {
                count++;  // checking # of characters
                seen=0;
        }
    }
```

```
        if (seen)
            count++;
        return (count);  /* returning No. of characters */
    }

int main() {

    int pid, word_count=0;   /* process id & word count */
    FILE *current_file = fopen("test.txt ","r");
    pid = vfork();

    if (pid == -1)           /* Error in forking process */
        exit(-1);
    else if (pid==0)         /* Child process */
    {
        word_count = countWords(current_file);
        printf("Child process found that number of words
                is : %i\n", word_count);
        exit(0);
    }
    else                     /* Parent process */
    {
        wait(NULL);
    }
    printf("Total Number of words: %i \n.", word_count);

}
```

PAGE NO.
DATE: / /

2.

(a) FCFS - Does not discriminate in favour of short processes. Process that request CPU first is allocated CPU first. Ready list is maintained as FIFO queue. Hence if a process with large CPU burst comes first, it is allocated the CPU until it releases CPU voluntarily irrespective of if all other upcoming processes are of shorter CPU burst.

(b) RR - Does not discriminate in favour of short processes. Ready queue is a circular queue. So processes are allocated CPU in the order in which they come in a circular manner for time slice. So it doesn't favor shorter processes in any way.

(c) Multi-level feedback Queue - discriminates in favour of short-processes. Long running processes move to lower priority queues which are executed only when all higher priority queues have completed execution and are empty. This delays the CPU allocation for long processes.

PAGE NO.
DATE: / /

(i)

3. (a) Given, 10 processes = $P_1, P_2 \ldots P_{10}$ [order in which they are present in ready queue]

Computation time of each loop = 50 msec

I/O operation takes 200 msec

time slice = 50 msec ⇒ 1 loop of a process $P_i$ can be computed in one time slice.

Scheduling Overhead = 3 msec

Following Round Robin Algorithm,

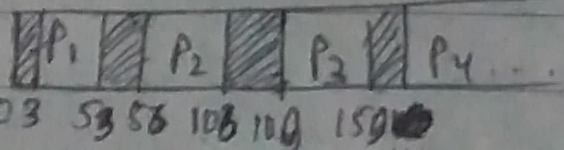
Response time for the 1st iteration = $(3 + 50)\,i$ $[i = 1, 2 \ldots 10]$

(3 ↑ scheduling overhead, 50 ↑ time slice)

Then 1st loop enters into I/O operation and $P_{iH}$ is scheduled and so on

∴ Response time for 1st iteration = $(53)\,i$ for $i^{th}$ process $i = [1, 10]$

ii) Gantt chart




$P_i$ leaves CPU at $(53)i$ for 1st loop

So 2nd iteration of $P_i$ starts after

$(53 \times 10) - 200$

$= 330$ ms

∴ Response time in the 1st and subsequent iteration for all processes = 330 ms.

(i)

(b) Time slice = 20 msec

For 1st iteration:

$$(\underbrace{10 \times 23}_{\downarrow} + 10 \times 23 + \underbrace{13}_{\downarrow}) \text{ msec}$$

10 = processes

23 = 20 + 3

20 ↑ time slice

3 ↑ overhead

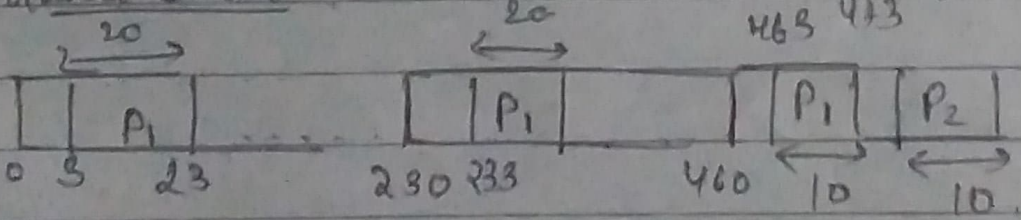
10 + 3 ↓ the iteration has only 10 msec left for computation

For ith process,

$$(10 \times 23 + 10 \times 23 + 13i) = (460 + 13i) \text{ msec}$$

∴ Response time in the first iteration = $(460 + 13i)$ msec for process $P_i$ (473, 486 ... ms)

(ii) Gantt chart




∴ 2nd process is ready 3 s after the 1st process finishes I/O similarly 3rd process is ready 26 sec after 1st process finishes I/O

PAGE NO.
DATE: / /

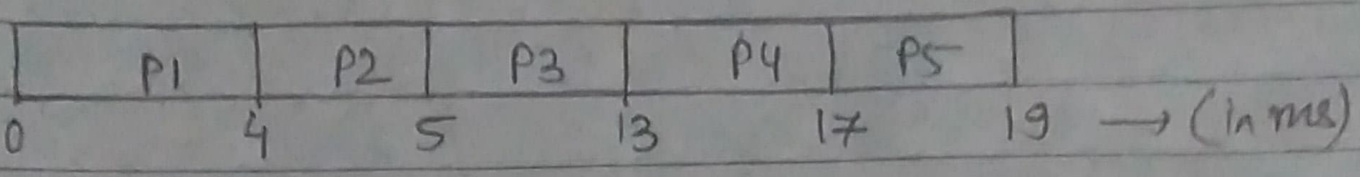
Finally, arrival time = $13(i-1)$ for $i=[1,10]$

arrival time.

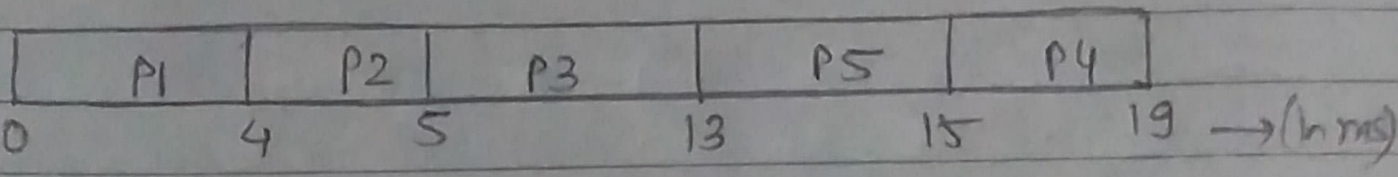
$443 - \underbrace{13(i-1)}_{} + 13(i-1)$

$= 443$ msec for all process.

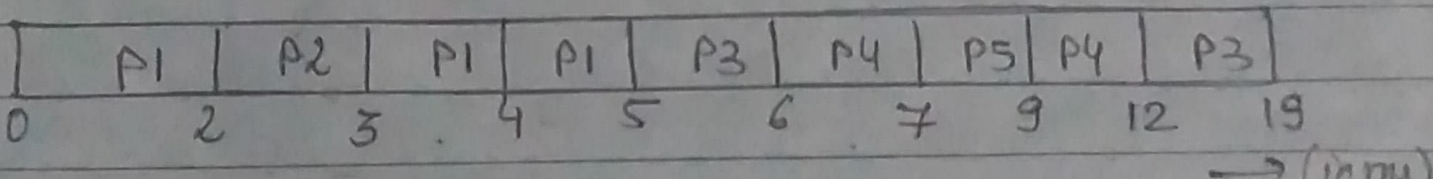
4. (a)(i) FCFS



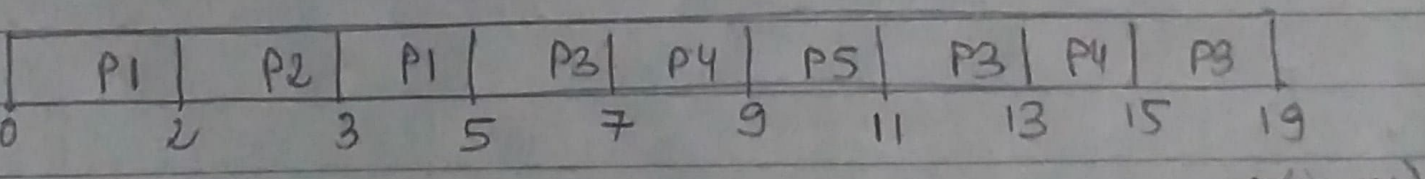

(ii) SJF




(iii) pre-emptive SJF




(iv) RR (with time quantum = 2)




(b) (i) For FCFS,

$$\text{Average turnaround time} = \frac{\underset{P_1}{(4-0)} + \underset{P_2}{(5-2)} + \underset{P_3}{(13-4)} + \underset{P_4}{(17-6)} + \underset{P_5}{(19-7)}}{5}$$

$$= \frac{39}{5} = 7.8 \text{ ms}$$

PAGE NO.
DATE / /

For SJFS, $P_1 = 4-0 = 4$ ms  $P4 = 19-6 = 13$ ms
$P2 = 5-2 = 3$ ms  $P5 = 15-7 = 8$ ms
$P3 = 13-4 = 9$ ms

Average turnaround time = $\frac{4+3+9+13+8}{5} = \frac{37}{5}$

$= 7.4$ ms

For pre-emptive SJFS,

$P_1 = 5-0 = 5$ ms  $P4 = 12-6 = 6$ ms
$P2 = 3-2 = 1$ ms  $P5 = 9-7 = 2$ ms
$P3 = 19-4 = 15$ ms

Average turnaround time = $\frac{5+1+15+6+2}{5} = \frac{29}{5} = 5.8$ ms

For RR,

$P_1 = 5-0$ ms $= 5$ ms ; $P4 = 15-6$ ms $= 9$ ms
$P2 = 3-2$ ms $= 1$ ms ; $P5 = 11-7$ ms $= 4$ ms
$P3 = 19-4$ ms $= 15$ ms

Average turnaround time = $\frac{5+1+15+9+4}{5} = \frac{34}{5} = 6.8$ ms

4.(b)(ii) For FCFS,

$P1 = 0-0 = 0$ ms, $P2 = 4-2 = 2$ ms, $P3 = 5-4 = 1$ ms

P4 = 13 - 6 = 7 ms
P5 = 17 - 7 = 10 ms

∴ Average waiting time = $\frac{0+2+1+7+10}{5}$ = 4 ms

For SJF,

P1 = 0 - 0 = 0 ms
P2 = 4 - 2 = 2 ms
P3 = 5 - 4 = 1 ms
P4 = 15 - 6 = 9 ms
P5 = 13 - 7 = 6 ms

∴ Average waiting time = $\frac{0+2+1+9+6}{5} = \frac{18}{5}$ = 3.6 ms

For preemptive SJF,

P1 = (3-2) + (0-0) = 1 ms
P2 = (2-2) = 0 ms
P3 = (12-6) + (5-4) = 7 ms
P4 = (9-7) + (6-6) = 2 ms
P5 = (7-7) = 0 ms

Average waiting time = $\frac{1+0+7+2+0}{5}$ = 2 ms

For RR,

P1 = (3-2) + (0-0) = 1 ms
P2 = (2-2) = 0 ms
P3 = (15-13) + (11-7) + (5-4) = 7 ms
P4 = (13-9) + (7-6) = 5 ms
P5 = (9-7) = 2 ms

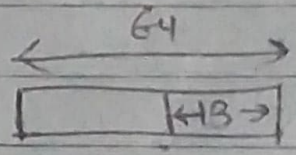
Average waiting time = $\frac{1+0+7+5+2}{5} = \frac{15}{5} = 3$ ms

5. (a) RAM = 128GB = $2^{37}$ bytes

Given, Page size = 8KB = $2^{13}$ bytes

∴ No. of frames in RAM (physical Memory) = $\frac{2^{37} \text{ bytes}}{2^{13} \text{ bytes}}$

$\left[\frac{\text{RAM size}}{\text{Page size}}\right] = 2^{24}$ frames




Given, 64 bit address space is used.

∴ Logical address space has $2^{64}$ byte memory locations.

∴ No. of entries in the page table = $\frac{2^{64}}{2^{13}}$ $\left\{\frac{\text{Logical space}}{\text{Page size}}\right\}$

$= 2^{51}$

Each page table entry contains frame no. of RAM.

As frame no. range from (0 to $2^{24}-1$) = 24 bits needed

∴ Each page table entry is of 24 bits = 3 bytes

∴ Total page table size = $3 \times 2^{51}$ bytes

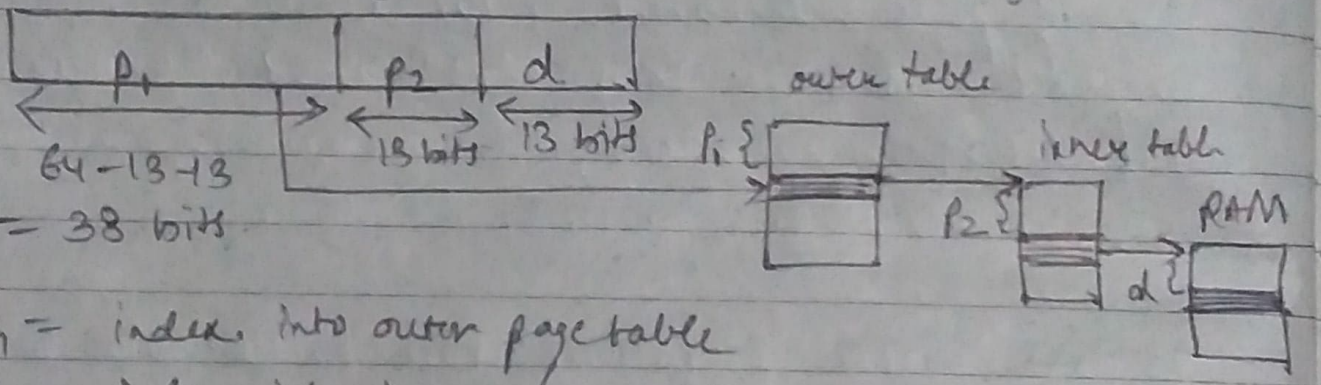
[ignoring valid-invalid bit and dirty bit]

PAGE NO.
DATE: / /

5. (b) RAM = 128 GB = $2^{37}$ bytes
Given page size = 8KB = $2^{13}$ bytes

∴ # of frames in RAM = $\frac{\text{RAM size}}{\text{Page size}} = \frac{2^{37}}{2^{13}} = 2^{24}$ frames.

logical address space = 64 bits
Dividing address space into $p_1$ (page number of outer table), $p_2$ (page number of inner table) and d (page offset)




$p_1$ = index into outer page table
$p_2$ = index into inner page table
d = offset of frames in RAM

∴ No. of entries in inner page table = $2^{13}$
Each entry size in inner page table = 24 bits
(∵ frame = $2^{24}$)
≈ 3 bytes

∴ Size of inner Page table = $(3 \times 2^{13})$ bytes = 24KB

∴ No. of entries in outer page table = $2^{38}$

PAGE NO
DATE / /

Each entry size in outer page table = 13 bits
(as inner page table has $2^{13}$ entries)
$\approx$ 2 bytes

$\therefore$ Size of outer page table = $2 \times 2^{38}$ bytes

$= 2^{39}$ bytes $=$ 512 GB.

$\therefore$ Total size of Outer and inner page tables $= (2^{39} + 3 \times 2^{13})$ bytes

$=$ 512 GB + 24 KB

$\approx$ 512 GB.

6. (a)

PAGE NO.
DATE: / /

Following Algorithms can be used to minimise page fault.

→ Least Frequently Used (LFU): Replace page having smallest count of number of references made to it. (Implemented in software level)

→ Most Frequently Used (MFU): Replace page that is referenced maximum number of times. As page with smallest count is yet to be used. (Implemented on software level)

→ Least Recently Used (LRU): Associate time of last use with each page and replace page that has not been used in the most amount of time (Implemented on software level)

→ Optimal Algorithm: Replace page that will not be used for the longest period of time. (Implemented on software level)

→ Thrashing: Process busy swapping pages in and out.
Minimising thrashing minimises page fault. (Software)

→ Locality Model: Allocate enough frames to a process to accomodate its current locality. Minimises thrashing, hence minimises page fault rate. (Software)

→ Working-Set Model: Allowing page replacement depending upon the working set value of a process (total number of pages referenced in the most recent working-set-window). If total demand of frames across all $P_i$ > memory size, thrashing occurs. So suspend or swap out one of the processes. (Software)

→ Page fault frequency: Page fault rate varies with the number of frames for process. Maintain rate between upper and lower bound (acceptable rate) to allow enough frames of the process to be in memory. (Software)

---

(b) → Dirty bit: Associate dirty bit with each page to know if page has been modified while in memory. Supports deciding if the page needs to be swapped out before replacing it as swapping out has overhead, thus decreasing time of page fault. (Software)

→ Swap Space: Using swap space avoids the access time from the disk directly. Data transfer rate is comparatively lower. Thus bringing pages from swap space decreases the extra overhead from the disk.
(Hardware implementation)

7.

Suppose process P caused a page fault, then

a) a trap is send to the operating system
b) Save the user registers and process state of P
c) determine that the interrupt was a page fault
d) Check that the page reference is legal and determine location of page on disk
e) Change state of process P to waiting
f) Issue a read from the disk and dwelling for free frame
g) If there is no free frame, Select page of another process Q as victim frame.
h) If the dirty bit of victim frame is 1, write frame to disk.
i) Now issue a read from the disk for desired page of P.
j) While P is waiting, allocate CPU to some other process
k) Receive an interrupt from the disk I/O subsystem as reading page is completed
l) Save the registers and process state of the others process that was running.
m) determine that the interrupt was from disk
n) Change the valid bit to 'i' for the victim page in the page table of Q.
o) Change the valid bit to 'v' for the page that caused page fault in the page table of P and assign victim frame no. to the entry.
p) Wait for the CPU to be allocated to P again
q) When P is allocated again to CPU, restore the user registers, process state and new page table of P and then resume the

PAGE NO.
DATE: / /

interrupted instruction that caused page fault.