2020

OPERATING SYSTEMS

18203023

SHRI KRISHNA DIDWANIA

UNIVERSITY COLLEGE DUBLIN | **IRELAND**

ASSIGNMENT 2

QUESTION 1

Exercise 1 *Consider a computer system using paging, where the address space of every process has a size of $C = 2^c$ bytes and the page size is $S = 2^s$ bytes. Each entry in the page table uses E bytes.*

1. *Calculate the number of pages of a process, and the size of a page table (in bytes).*
2. *Assume that the space wasted by a process in main memory is defined as the sum of the size of its page table plus the average internal fragmentation due to that process (i.e., one full page on average, considering that half of a page containing the heap and half of a page containing the stack are empty on average). Obtain the optimum page size, i.e., the page size that minimises the waste of space due to a process. Hint: put the wasted space as a function of the page size, and then optimise with respect to it.*
3. *Calculate the optimum page size assuming that $C = 4$ MiB and $E = 4$ B.*

SOLUTION

Information provided:

Process size (C)= 2^c bytes

Page size (S)=2^s bytes

Page table entry size=E bytes

- Part 1
  - **Number of pages** = process size / page size
    = 2^c / 2^s

= **$2^{(c-s)}$**

Since number of pages=number of entries in page table

- **Size of page table** = size of one entry x number of pages

  = **$E \times 2^{(c-s)}$ bytes**

- Part 2

Important point noted

- Internal fragmentation is only half the page.

Space wasted by a process in an overhead (S)= (size of page table) + (size of page/2)

Let the space wasted be denoted by **S** and **P** be the page size.

Now for minimum wastage,

$d(S) / d(P) = 0$

d(size of page table) + (size of page/2)) / d(size of page) = 0

On differentiating we get,

Page size = $\sqrt{(2 \times \text{Process size} \times \text{Page table entry size})}$

Which is the optimal size which minimizes wastage = $\sqrt{(2 \times 2^c \times E)}$

- Thus the **optimal size** is **$\sqrt{(2 \times 2^c \times E)}$**

- Part 3

- **Optimal page size** = $\sqrt{(2 \times 4\text{MiB} \times 4\text{B})}$

  = $\sqrt{(2 \times 2^{22}\ \text{B} \times 4\text{B})}$

  = $\sqrt{(2 \times 2^{22}\ \text{B} \times 2^2)}$

  = $\sqrt{(2^{25})}\ \text{B}$

Thus the **page size** is **5792.61875148 ~ 6KB** bytes.

QUESTION 2

Exercise 2 *Consider the following precedence relationships between processes P_1, P_2, P_3, P_4, P_5 and P_6:*

- P_1 *before* P_2 *and* P_3
- P_2 *before* P_4 *and* P_5
- P_3 *before* P_5
- P_6 *after* P_3 *and* P_4

where "P_i before P_j" means that the execution of process P_i must be completed before the execution of process P_j.

Define and initialise all necessary shared semaphores, and write pseudocode for all six processes using these semaphores in such a way that the precedence relationships above are always observed when the processes are run concurrently in a multiprogrammed operating system.

Semaphores

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.

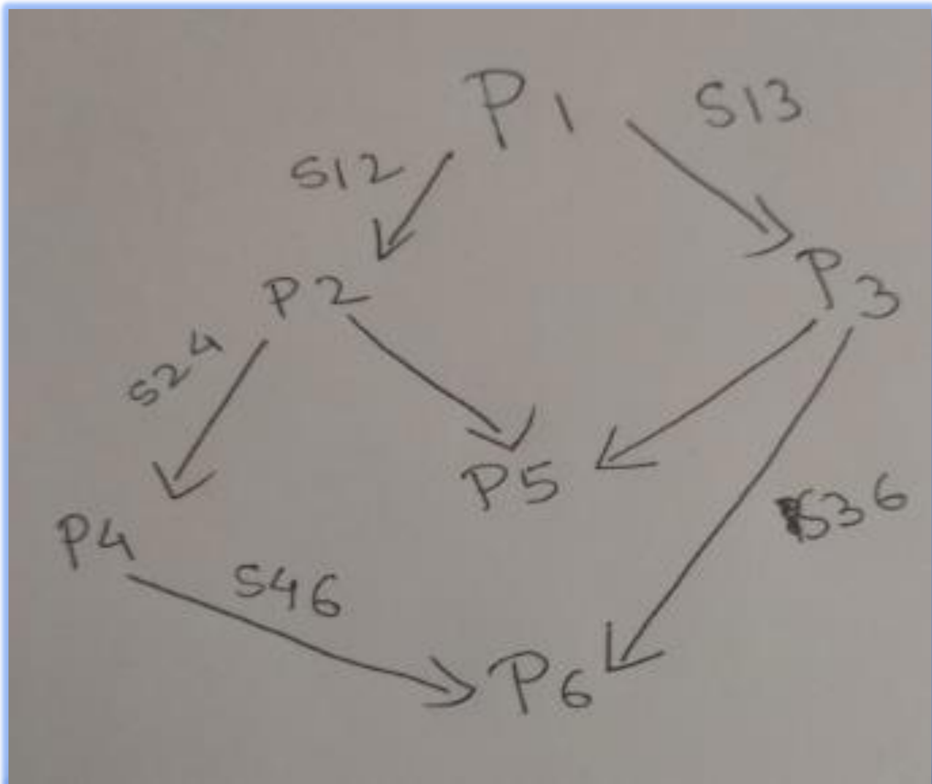
```
P(Semaphore s){
   while(S == 0);   /* wait until s=0 */
  s=s-1;
}

V(Semaphore s){
         s=s+1;
}
```

Note that there is
Semicolon after while.
The code gets stuck
Here while s is 0.

SAB represents semaphores shared between A & B

- S12 = 0 : S13 = 0   // Shared semaphores between 1 & 2, 1 & 3
- S24 = 0 : S25 = 0   // Shared semaphores between 2 & 4, 2 & 5
- S35 = 0 :S36=0   // Shared semaphores between 3 & 5, 3 &6
- S46 = 0 :   // Shared semaphores between 4 & 6
- S56 = 0   // Shared semaphores between 5 & 6



Execution process:

P1	P2	P3	P4	P5	P6
//Since there is no restriction on P1 for entering, we don't have a semaphore to check the entrance of P1 so it can enter the CS directly	P(S12);	P(S13);	P(S24);	P(S25); P(S35);	P(S46); P(S36);
	// Checks if the process P2 is free to enter i.e. if (S12 = 1) and makes S12 = 0 after P2 enters.	// Checks if the process P3 is free to enter i.e. if (S13 = 1) and makes S13 = 0 after P3 enters.	// Checks if the process P4 is free to enter i.e. if (S24 = 1) and makes S24 = 0 after P4 enters.	// Checks if the process P5 is free to enter (i.e. will execute only after processes 2 and 3 have executed) i.e. if (S25 = 1 and S35 = 1) and makes S25 = 0 and S35 = 0 after P5 enters.	// Checks if the process P6 is free to enter (i.e. will execute only after processes 4 and 3 have executed) i.e. if (S46 = 1 and S36 = 1) and makes S46 = 0 and S36 = 0 after P6 enters.
// CS	*// CS*	*// CS*	*// CS*	*// CS*	*// CS*
V(S12);	V(S24);	V(S35);	V(S46);		
//Changes the value of semaphore S12 to 1, hence signaling P2	//Changes the value of semaphore S24 to 1, hence signaling P4	//Changes the value of semaphore S35 to 1, hence signaling P5	//Changes the value of semaphore S46 to 1, hence signaling P6	*// The process P5 does not signal any other processes since there is no condition on P5 to execute before a certain process.*	*// The process P6 does not signal any other processes since there is no condition on P6 to execute before a certain process.*
V(S13);	V(S25);	V(S36);			
//Changes the value of semaphore S13 to 1, hence signaling P3	//Changes the value of semaphore S25 to 1, hence signaling P5	//Changes the value of semaphore S36 to 1, hence signaling P6			

QUESTION 3

Exercise 3 *Consider m updater processes and n browser processes concurrently accessing a database in a multiprogrammed operating system, using the pseudocode in the figure below:*

```
int count = 0;
semaphore s_db(1);          } shared
semaphore s_count(1);

void browser()                  void updater()
{                               {
   P(s_count);                     P(s_db);
   count++;
   if (count==1)                   // update db
      P(s_db)
   V(s_count);                     V(s_db);
                                }
   // browse db

   P(s_count);
   count− −;
   if (count==0)
      V(s_db);
   V(s_count);
}
```

Briefly discuss the following questions about the solution above (do not just answer "yes" or "no").

1. *If an updater is updating, in what semaphore(s) do browser processes block?*
2. *If a browser is browsing, can other browsers be prevented from browsing?*
3. *How many updater processes can concurrently update the database?*
4. *Does the same browser process which executes P(s_db) also execute V(s_db)?*
5. *Is the solution given free of deadlock and starvation?*
6. *What would happen if the only semaphore were s_db? (that is, if we would delete all lines related to s_count in the pseudocode).*

SOLUTION

Part 1

The first browser process **blocks** in semaphore **s_db** and thereafter **all** the **other browser** processes are blocked in the **semaphore s_count**.

Part 2

No, the other browsers cannot be prevented from browsing since while browsing , the value of semaphore s_count becomes 1 , hence it is available. So any number of processes can browse while one process is browsing already.

Part 3

Only one updater process can update the database concurrently

Let there be 2 updater process P1 and P2, Let s_db=1

We assume that P1 is being updated first. Now s_db=0. Now we try to bring another process P2 to update database. Since s_db=0; hence the process P2 gets blocked. Hence, it is proved at a time only one person can update the database.

Part 4

No, it is not necessary that the same browser process that executes P(s_db) will execute V(s_db). Since, the first process to start browsing may not be the last process that browses the database.

If there is only one browser process, only then it is guaranteed that it execute both P(s_db) and V(s_db).

Part 5

Yes, the solution is free of deadlock. But the solution is not starvation free. Since any number of processes can browse together, the writer processes may starve if browser processes keep on coming.

There is no deadlock because no processes are blocked

Part 6

Suppose there are two browser processes:

```
Process P1
While(1)
{
Count++;            1
If(count==1)        2
P(s_db);            3
//browse
 Count --;          4
If(count==0)        5
V(s_db);            6
}
```

```
Process P2
While(1)
{
Count++;                a
If(count==1)            b
P(s_db);                c
//browse
 Count --;              d
If(count==0)            e
V(s_db);                f

}
```

Let there be another process P3------updater process

```
Process P3
While(1)
{
P(s_db);                A
//updating
V(s_db);                B
}
```

Execution:- count-->0→1→2 s_db→1→0

Process P3 is updating

1--------a----------b-----------A--------

Context Switch

Now the process P2 is browsing

Again context switch to P3

From the above execution it is seen that both browser and updater process are working at the same time which is a wrong situation.

Since the s_count semaphore is removed, the count value is updated and P(s_db) statement is not executed in the browser process. Hence the updater process is able to execute at the same time browser process is executing.