



| | | | | |
|---------|--|-----------|--|-------|
| SURNAME | | NAME | | Group |
| ID | | Signature | | |

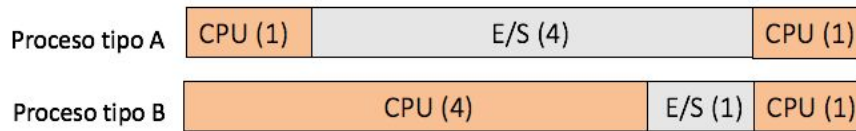
- Keep the exam sheets stapled.
- Write your answer inside the reserved space.
- Use clear and understandable writing. Answer briefly and precisely.
- The exam has 7 questions, everyone has its score specified.

1. Considering the following data stored on a process PCB, explain for everyone if it changes or not from the process start to its end and what is the change cause.

(1,2 points = 0,3+0,3+0,3+0,3)

| | |
|----|---|
| 1a | PID It doesn't change. PID is an attribute that has a unique value for every process (process identification) and it remains the same along the whole process lifetime. |
| 1b | PPID It can change if the parent process ends before, if so the new parent will be INIT and PPID becomes 1. |
| 1c | Copy of program counter and CPU registers It changes every time there is an exception. The OS keeps a copy of these CPU registers on the PCB, to use them when the process context has to be restored. |
| 1d | State It changes. Here it is indicated the actual state of the process, so it changes every time there is an state change between: waiting, ready, execution, suspended, etc |

2. Let's consider a system with only one CPU and I/O made by one hard disk, that allow a **maximum concurrency of 2 processes**. Two type of processes can arrive to this system, they alternate CPU and I/O bursts as described on the following picture (burst length is indicated on parenthesis):



Ask the following questions as much concisely as possible:

(1,7 points = 0,6+0,6+0,5)

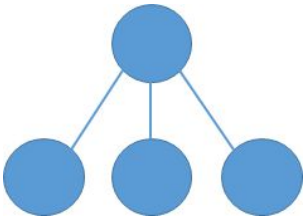
| | |
|------------------|--|
| <p>2a</p> | <p>Describe what is a CPU bound process and what is an I/O bound process, and tell to what kind correspond type A processes and type B processes</p> <p>CPU bound process: a process which its CPU bursts are longer than its I/O bursts, so its execution time is limited by the CPU speed.</p> <p>I/O bound process: a process which its I/O bursts are longer than its CPU bursts, so its execution time is limited by the I/O speed.</p> <p>Type A process: I/O bound</p> <p>Type B process: CPU bound</p> |
| <p>2b</p> | <p>Indicate what process combination and arrival order gives the maximum performance (throughput) and the maximum CPU utilization. Justify your answer graphically and compute the throughput and CPU utilization.</p> <p>Note. Running processes can be of the same type A or B</p> <p>The optimal combination is 1 type A process and 1 type B process, entering first type A into the CPU. In that way while A is doing I/O, B can do CPU, and then I/O and CPU activities are overlapped.</p> <p>Throughput is 2/7 and CPU utilization is 100%.</p> |
| <p>2c</p> | <p>If the hard disk is replaced by an SSD reducing I/O burst time in 1/4, indicate what mix of process types will provide the maximum throughput.</p> <p>Reducing I/O cost, the most benefited processes are the I/O bound ones. Particularly, type A processes that have one 4 ut I/O burst, it is reduced to 1 ut. So the best combination now is two type A processes that provide a throughput of 2/4 and a CPU utilization of 100%.</p> |

3. Given the following programs:

| | |
|---|--|
| F.c code: <pre> 1 #include <stdlib.h> 2 #include <stdio.h> 3 int main(int argc, char * argv[]){ 4 int pid,i,status; 5 for(i=0;i<3;i++){ 6 pid=fork(); 7 if (pid==0){ 8 printf("exit\n"); 9 exit(i); 10 } 11 } 12 i=0; 13 while(wait(&status)>0) i++; 14 printf("wait(%d)\n",i); 15 }</pre> | E.c code: <pre> 1 #include <unistd.h> 2 int main(int argc, char * argv[]){ 3 execl("./F","F",NULL); 4 execl("./F","F",NULL); 5 }</pre> |
|---|--|

Suppose that both are compiled into E and F on the default folder.

(1,6 points = 0,4+0,4+0,4+0,4)

| | |
|-----------|---|
| 3a | <p>In all, how many processes creates the command ./F? Draw the process tree.</p> <p>Four processes in all: one parent and three children</p>  <pre> graph TD A(()) --- B(()) A --- C(()) A --- D(()) </pre> |
| 3b | <p>What is the command ./F output on the terminal?</p> <p>The output is made of the following four lines:</p> <pre> exit exit exit wait(3)</pre> |
| 3c | <p>From processes created with command ./F, how many become orphan? Explain your answer.</p> <p>None, all processes that create children wait for all of them to end</p> |
| 3d | <p>How many processes creates command ./E? Explain your answer</p> <p>The same as in item 3a). The first call to exec changes executable E by F</p> |

4. We intend to analyze short term schedulers for the operating system running on a computer. The selected schedulers are: SJF (Shortest-Job-First), SRTF (Shortest-Remaining-Time-First), y RR (Round-Robin) with 1 ut *quantum* ($q = 1$). The analysis relies on the mean turnaround time, mean waiting time and CPU utilization of the following processes:

| Process | Arrival time | CPU and I/O bursts |
|---------|--------------|-----------------------|
| A | 0 | 4 CPU + 1 I/O + 1 CPU |
| B | 1 | 2 CPU + 2 I/O + 1 CPU |
| C | 4 | 1 CPU |

Fill up the following execution tables for every scheduler and compute the mean turnaround time, the mean waiting time and the CPU utilization. In case of simultaneous events consider the following order: 1) new job arrival, 2) job ending, 3) leaving suspended state y 4) quantum ending. Which one will be the best scheduler considering independently the mean turnaround time, the mean waiting time and the CPU utilization.

(1,7 points = 0,5 + 0,5 + 0,5+ 0,2)

4a) SJF (Shortest-Job-First)

| T | Ready | CPU | I/O Queue | I/O | Event |
|----|--------|-----|-----------|-----|-----------|
| 0 | (A) | A | | | A arrives |
| 1 | B | A | | | B arrives |
| 2 | B | A | | | |
| 3 | B | A | | | |
| 4 | B, (C) | C | (A) | A | C arrives |
| 5 | B, (A) | A | | | C ends |
| 6 | (B) | B | | | A ends |
| 7 | | B | | | |
| 8 | | | (B) | B | |
| 9 | | | | B | |
| 10 | | B | | | |
| 11 | | | | | B ends |

| | A | B | C | Mean |
|-----------------|-----------------|----|---|------------------|
| Turnaround time | 6 | 10 | 1 | $17/3 = 5,67$ ut |
| Waiting time | 0 | 5 | 0 | $5/3 = 1,67$ ut |
| CPU utilization | $9/11 = 81.8\%$ | | | |

4b) SRTF (Shortest-Remaining-Time-First)

| T | Ready | CPU | I/O Queue | I/O | Event |
|----|--------|-----|-----------|-----|-----------|
| 0 | (A) | A | | | A arrives |
| 1 | A, (B) | B | | | B arrives |
| 2 | A | B | | | |
| 3 | (A) | A | (B) | B | |
| 4 | A, (C) | C | | B | C arrives |
| 5 | A, (B) | B | | | C ends |
| 6 | (A) | A | | | B ends |
| 7 | | A | | | |
| 8 | | | (A) | A | |
| 9 | (A) | A | | | |
| 10 | | | | | A ends |

| | A | B | C | Mean |
|-----------------|---------------|---|---|--------------------------|
| Turnaround time | 10 | 5 | 1 | $16/3 = 5,33 \text{ ut}$ |
| Waiting time | 4 | 0 | 0 | $4/3 = 1,33 \text{ ut}$ |
| CPU utilization | $9/10 = 90\%$ | | | |

4c) RR (Round-Robin) $q = 1 \text{ ut}$

| T | Ready | CPU | I/O Queue | I/O | Event |
|----|--------|-----|-----------|-----|-----------|
| 0 | (A) | A | | | A arrives |
| 1 | A, (B) | B | | | B arrives |
| 2 | B, (A) | A | | | |
| 3 | A, (B) | B | | | |
| 4 | C, (A) | A | (B) | B | C arrives |
| 5 | A, (C) | C | | B | |
| 6 | B, (A) | A | | | C ends |
| 7 | (B) | B | (A) | A | |
| 8 | (A) | A | | | B ends |
| 9 | | | | | A ends |
| 10 | | | | | |
| 11 | | | | | |
| 12 | | | | | |
| 13 | | | | | |

| | A | B | C | Mean |
|-----------------|------|---|---|-----------------------|
| Turnaround time | 9 | 7 | 2 | $18/3 = 6 \text{ ut}$ |
| Waiting time | 3 | 2 | 1 | $6/3 = 2 \text{ ut}$ |
| CPU utilization | 100% | | | |

4d) Scheduler selection

| Relying index | Best scheduler | Best value |
|-----------------|----------------|--------------------------|
| Turnaround time | SRTF | $16/3 = 5,33 \text{ ut}$ |
| Waiting time | SRTF | $4/3 = 1,33 \text{ ut}$ |
| CPU utilization | RR | 100% |

5. Program ThreadsAdd.c from the threads lab session is shown below. It does the addition of a bidimensional matrix.

| | |
|---|--|
| <pre> 1 #include <stdio.h> 2 #include <pthread.h> 3 4 #define DIMROW 1000000 5 #define NUMROWS 20 6 7 typedef struct row{ 8 int vector[DIMROW]; 9 long addition; 10 } row; 11 12 struct row matrix[NUMROWS]; 13 long total_addition=0; 14 15 void *AddRow(void *ptr) 16 { 17 int k; 18 row *fi; 19 fi = (row *)ptr; 20 21 fi->addition=0; 22 for(k=0;k<DIMROW;k++) 23 fi->addition += exp((k*(fi->vector[k])+(k+1)* (fi->vector[k]))/ (fi->vector[k]+2*k))/2; 24 //APDO.D 25 } </pre> | <pre> 26 int main() 27 { 28 int i,j; 29 pthread_t threads[NUMROWS]; 30 pthread_attr_t atrib; 31 32 // Vector elements are initialized to 1 33 for(i=0;i<NUMROWS;i++) 34 for(j=0;j<DIMROW;j++) 35 matrix[i].vector[j]=1; 36 37 // Thread attributes initialization 38 pthread_attr_init(&atrib); 39 40 for(i=0;i<NUMROWS;i++){ 41 pthread_create(&threads[i],&atrib, 42 AddRow, (void *)&matrix[i]); 43 } 44 45 for(i=0;i<NUMROWS;i++) 46 pthread_join(threads[i],NULL); 47 48 for(i=0;i<NUMROWS;i++) 49 total_addition += matrix[i].addition; 50 printf("Total addition is: %ld \n", 51 total_addition); 52 pthread_exit(0); 53 } </pre> |
|---|--|

Answer the following questions about program ThreadsAdd.c. In all questions the changes are done on the initial code and we suppose that the execution is done on a multicore processor.

(1,2 points = 0,3 + 0,3 + 0,3 + 0,3)

| | |
|-----------|--|
| 5a | <p>What will be the effect on the final result got if lines 45 and 46 are removed?</p> <p>The result may not be correct because the total sum is calculated and printed without waiting for all the threads to finish obtaining the partial sum of each row.</p> |
| 5b | <p>What will be the final result and the execution time got if lines 45, 46 and 51 are removed?</p> <p>All threads would end quickly because the main process would end without waiting for the rest of the threads to finish. The result would be incorrect. Very short execution time.</p> |

| | |
|-----------|---|
| 5c | <p>What will be the final result and the execution time got if lines 45 and 46 are removed and the sentence <code>"pthread_join(threads[i],NULL);"</code> is appended at line 42?</p> <p>Threads that add every row would not run in parallel, they would wait for each row completion before starting the next one. The result would be correct but the execution time would be greater, equivalent to sequential execution.</p> |
| 5d | <p>If lines 48 and 49 are removed and the sentence <code>"total_addition += fi->addition;"</code> is appended at line 24, will be the result got correct?</p> <p>The result may be incorrect. A race condition would appear because the threads would access the <code>total_addition</code> shared variable without any critical section access protocol.</p> |

6. Given functions add and sub seen on lab sessions, answer the following questions:

NOTE. Consider that before declaring the former functions the following constants and variables are declared:

```
#define REPEAT 20000000
long int V = 100;
int Key = 0;
```

| | | | |
|----|--|----|--|
| 1 | void *add (void *argument){ | 15 | void *sub (void *argument) { |
| 2 | long int count; | 16 | long int count; |
| 3 | long int aux; | 17 | long int aux; |
| 4 | | 18 | |
| 5 | for (count=0; count<REPEAT; count++) { | 19 | for (count=0; count<REPEAT; count++) { |
| 6 | | 20 | |
| 7 | V = V + 1; | 21 | V = V - 1; |
| 8 | | 22 | |
| 9 | } | 23 | } |
| 10 | | 24 | |
| 11 | printf("--> End ADD (V=%ld) \n", V); | 25 | printf("--> End SUB (V=%ld)\n", V); |
| 12 | pthread_exit(0); | 26 | pthread_exit(0); |
| 13 | } | 27 | } |
| 14 | | 28 | |
| 29 | int main (void) { | | |
| 30 | pthread_t threadAdd, threadSub, | | |
| 31 | pthread_attr_t attr; | | |
| 32 | | | |
| 33 | pthread_attr_init(&attr); | | |
| 34 | pthread_create(&threadAdd, &attr, add, NULL); | | |
| 35 | pthread_create(&threadSub, &attr, sub, NULL); | | |
| 36 | | | |
| 37 | pthread_join(threadAdd, NULL); | | |
| 38 | pthread_join(threadSub, NULL); | | |
| 39 | | | |
| 40 | fprintf(stderr, "-----> FINAL VALUE: V = %ld\n\n", V); | | |
| 41 | exit(0); | | |
| 42 | } | | |

(1,6 points = 0,4 + 0,4 + 0,4 + 0,4)

| | |
|-----------|--|
| 6a | <p>What is a critical section? If there are critical sections in the former code, tell on what line numbers they are and on what lines the input protocol and the output protocol to protect them have to be added.</p> <p>The critical section is the part of the thread code where shared variables are accessed In the code the critical sections are in lines 8 and 22</p> <p>INPUT PROTOCOL</p> <p>lines 7 y 21</p> <p>OUTPUT PROTOCOL</p> <p>lines 9 y 23</p> |
|-----------|--|

| | |
|-----------|---|
| 6b | <p>If we protect the critical sections using basic Test&Set hardware solution, write the code that implements the input protocol and the output protocol.</p> <p>INPUT PROTOCOL</p> <pre>while(test_and_set(&Key));</pre> <p>OUTPUT PROTOCOL</p> <pre>Key = 0;</pre> |
| 6c | <p>Explain if the basic Test&Set hardware solution, the one used before, comply with the three conditions of the critical section access protocols.</p> <p>Hardware basic solutions based on Test & Set comply with the conditions of mutual exclusion and progress, since being an atomic operation is only allowed that a process accesses the key variable to change its value, therefore no more than one process can access the critical section at any given time. If there is no one in the critical section the key variable will be zero, and therefore as soon as a thread tries to access the critical section, it can do it.</p> <p>On the other hand, the limited waiting condition may not be met, because the decision on which process accesses the critical section is not on the protocol, but on the scheduler and if it is not fair, limited waiting may not be met.</p> |
| 6d | <p>Explain the relation between basic Test&Set hardware solution and active (busy) waiting. Under what conditions is recommended to use active waiting on the critical section access protocols?</p> <p>Hardware basic solutions based on Test & Set use active waiting, that is empty loops, to check the value of Key variable, and thus the ability to access the critical section.</p> <p>Active waiting can be not appropriate depending on the scheduling policy, particularly if there are priorities it can produce deadlock if the thread on the empty loop has more priority than the one that is executing the critical section.</p> <p>If scheduling is not a problem then active waiting is better than event waiting when the critical section is short and the number of competing threads is low, so the probability of having to wait for entering the critical section is very low.</p> |

7. Given three semaphores: A, B and C initialized as: A=0, B=1, C=0; there are 3 processes that run concurrently and perform the following accesses to the semaphores:

| Process 1 | Process 2 | Process 3 |
|---|---|--|
| . . P(A); . . . V(C); | . . P(B); . . . V(A); V(B); | . . P(B); P(C); . . V(B); |

(1 point = 0,5 + 0,5)

| 7a | <p>Indicate one sequence of P and V operations of the three processes that allow all of them to end.</p> <table> <tr> <th>PROCESS</th><th>SEMAPHORE OPERATION</th></tr> <tr> <td>Process 1</td><td>P (A)</td></tr> <tr> <td>Process 2</td><td>P (B)</td></tr> <tr> <td>Process 2</td><td>V (A)</td></tr> <tr> <td>Process 2</td><td>V (B)</td></tr> <tr> <td>Process 1</td><td>V (C)</td></tr> <tr> <td>Process 3</td><td>P (B)</td></tr> <tr> <td>Process 3</td><td>P (C)</td></tr> <tr> <td>Process 3</td><td>V (B)</td></tr> </table> | PROCESS | SEMAPHORE OPERATION | Process 1 | P (A) | Process 2 | P (B) | Process 2 | V (A) | Process 2 | V (B) | Process 1 | V (C) | Process 3 | P (B) | Process 3 | P (C) | Process 3 | V (B) |
|-----------|--|---------|---------------------|-----------|-------|-----------|-------|-----------|-------|-----------|-------|-----------|-------|-----------|-------|-----------|-------|-----------|-------|
| PROCESS | SEMAPHORE OPERATION | | | | | | | | | | | | | | | | | | |
| Process 1 | P (A) | | | | | | | | | | | | | | | | | | |
| Process 2 | P (B) | | | | | | | | | | | | | | | | | | |
| Process 2 | V (A) | | | | | | | | | | | | | | | | | | |
| Process 2 | V (B) | | | | | | | | | | | | | | | | | | |
| Process 1 | V (C) | | | | | | | | | | | | | | | | | | |
| Process 3 | P (B) | | | | | | | | | | | | | | | | | | |
| Process 3 | P (C) | | | | | | | | | | | | | | | | | | |
| Process 3 | V (B) | | | | | | | | | | | | | | | | | | |
| 7b | <p>Indicate one sequence of P and V operations of the three processes that end with a deadlock.</p> <table> <tr> <th>PROCESS</th><th>SEMAPHORE OPERATION</th></tr> <tr> <td>Process 3</td><td>P (B)</td></tr> <tr> <td>Process 3</td><td>P (C)</td></tr> <tr> <td>Process 2</td><td>P (B)</td></tr> <tr> <td>Process 1</td><td>P (A)</td></tr> </table> | PROCESS | SEMAPHORE OPERATION | Process 3 | P (B) | Process 3 | P (C) | Process 2 | P (B) | Process 1 | P (A) | | | | | | | | |
| PROCESS | SEMAPHORE OPERATION | | | | | | | | | | | | | | | | | | |
| Process 3 | P (B) | | | | | | | | | | | | | | | | | | |
| Process 3 | P (C) | | | | | | | | | | | | | | | | | | |
| Process 2 | P (B) | | | | | | | | | | | | | | | | | | |
| Process 1 | P (A) | | | | | | | | | | | | | | | | | | |