**Information Systems Engineering  
ISE302-Operating Systems  
Final Exam**

Asst. Prof. Gökhan Seçinti, R.A. Çağatay Koç

|  |  |
| --- | --- |
| Name: | Sarp Ozdemir |
| ID: | 150180207 |

**Q1)(15pt)** A process creates 3 child process and prints its own PID and children’s PIDs before termination. Every child process prints their creation order along with its own and its parent’s PIDs before termination.

Write the program following the behavior described above in C by using necessary Linux system calls. In addition, give an example output and briefly explain it.

**#include <stdio.h>**

**#include <unistd.h>**

**#include <sys/wait.h>**

**#include <stdlib.h>**

**int main()**

**{**

**for (int i=0;i<3;i++)**

**if (!fork())**

**{**

**printf("child: %d parent: %d order: %d \n",getpid(), getppid(), i+1);**

**exit(0);**

**}**

**for (int i=0;i<3;i++) wait(0);**

**return 0;**

**}**

**Example output:**

**child: 1201 parent: 1198 order: 3**

**child: 1200 parent: 1198 order: 2**

**child: 1199 parent: 1198 order: 1**

**Every child uses the pid and ppid functions to get the ids of themselves and their parents. They also use the i inherited from the parent process to determine the creation order of the processes which the execution belongs to. The order starts from one and 1 means that the process is the first child that has been created.**

**Q2)(20pt)** A roller coaster is able to serve C passengers at one ride. Passengers call *embark()* and *disembark()* functions to get on and off the roller coaster respectively. In addition, before any passenger gets on the roller coaster, the roller coaster should call *new\_ride()* function. Similarly, before any passenger gets off the roller coaster, it should call *end\_of\_ride()* function. The roller coaster starts it ride only when at full capacity (C passengers).

Write the pseudocode for passenger and roller coaster entities and explain the mutex and synchronization mechanisms you use clearly.

**int entering=0,exiting=0;**

**Mutex enter\_m = mutex(1);**

**Mutex exit\_m = mutex(1);**

**Mutex full = mutex(0);**

**Mutex empty = mutex(0);**

**Sempahore boarders= semaphore(0);**

**Semaphore unboarders = semaphore(0);**

**Void coaster()**

**{**

**New\_ride();**

**Boarders.signal(C);**

**Full.wait(1);**

**End\_of\_ride();**

**Unboarders.signal(c);**

**Empty.wait(1);**

**}**

**Void passenger()**

**{**

**Boarders.wait(1);**

**Embark();**

**Enter\_m.wait(1);**

**İf (++entering==C)**

**{**

**Entering=0;**

**Full.signal(1);**

**}**

**Enter\_m.signal(1);**

**Unboarders.wait(1);**

**Disembark();**

**Exit\_m.wait();**

**İf (++exiting == C)**

**{**

**Exiting=0;**

**Empty.signal(1)**

**}**

**Exit\_m.signal();**

**}**

**The enter\_m and exit\_m mutexes are mutexes which are used to keep the counters “entering” and “exiting” safe, these mutexes avoid situations such as two passengers incerementing one of the counters at the same time and not getting in the condition block because of this. The boarders and unboarders semaphores keep track of how many empty spaces are left inside and how many people are left inside who has to leave the coaster respectively. The full and empty mutexes are used to stop the coaster until the C’th passanger embarks which then leads to the if block inside the passenger thread to signal the coaster to start the ride.**

**Q4) (15pt)** Four different types of resources (A, B, C, D) are shared by five processes. The total number of resources that the processes can request during their entire operation and the resources they hold at a certain moment of their execution are given below.

**MAXIMUM NEED CURRENTLY ALLOCATED AVAILABLE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Process** | **A** | **B** | **C** | **D** |
| **P1** | 1 | 0 | 3 | 2 |
| **P2** | 2 | 1 | 1 | 1 |
| **P3** | 1 | 1 | 4 | 0 |
| **P4** | 1 | 0 | 2 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Process** | **A** | **B** | **C** | **D** |
| **P1** | 1 | 0 | 5 | 4 |
| **P2** | 7 | 2 | 1 | 3 |
| **P3** | 1 | 1 | 5 | 0 |
| **P4** | 3 | 5 | 2 | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **C** | **D** |
| 4 | 3 | 1 | 1 |

1. Calculate the remaining request matrix ii) Calculate the number of resources available at the beginning.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Process** | **A** | **B** | **C** | **D** |
| **P1** | 0 | 0 | 2 | 2 |
| **P2** | 5 | 1 | 0 | 2 |
| **P3** | 0 | 0 | 1 | 0 |
| **P4** | 2 | 5 | 0 | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **C** | **D** |
| 9 | 5 | 11 | 5 |

iii) Is this state of the system safe? If it is safe, give a sample resource assignment sequence that will allow all processes to terminate.

**This state is not safe. We can start by giving resources to p3 and freeing its resources but then we wont have enough reouces for any other processes.**

**Q4) (20pt)** The arrival times and execution periods and priorities of a set of processes are given below. For each of the following scheduling algorithms clearly state the execution sequence of the processes.

| **Process** | **Arrival Time** | **Exec. time** | **Priority** |
| --- | --- | --- | --- |
| P1 | 0 | 10 | 2 |
| P2 | 2 | 8 | 1 |
| P3 | 3 | 3 | 3 |
| P4 | 10 | 4 | 2 |
| P5 | 12 | 1 | 3 |
| P6 | 15 | 4 | 1 |

1. **FIFO**:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | X | X | X | X | X | X | X | X | X | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P2 |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X |  |  |  |  |  |  |  |  |  |  |  |  |
| P3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X |  |  |  |  |  |  |  |  |  |
| P4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X |  |  |  |  |  |
| P5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |
| P6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X |
| t | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |

1. **SJF:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | x | x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | x | x | x | x | x | x | x | x |
| P2 |  |  | x |  |  |  | x | x | x | X |  |  |  |  |  |  |  |  |  | x | x | x |  |  |  |  |  |  |  |  |
| P3 |  |  |  | x | x | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P4 |  |  |  |  |  |  |  |  |  |  | x | x |  | x | x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P5 |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | x | x | x | X |  |  |  |  |  |  |  |  |  |  |  |
| t | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |

1. **Shortest remaing time first:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | x | x | x |  |  |  | x | x | x | x | x | x | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | x | x | x | x | x | x | x | x |
| P3 |  |  |  | x | x | x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | x | x | x | X |  |  |  |  |  |  |  |  |  |  |  |  |
| P5 |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | x | x | x | X |  |  |  |  |  |  |  |  |
| t | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |

1. **Round robin (q=4):**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | x | X | x | x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | x | x | x | X |  |  |  |  | x | x |
| P2 |  |  |  |  | x | x | x | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | x | x | x | x |  |  |
| P3 |  |  |  |  |  |  |  |  | x | x | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P4 |  |  |  |  |  |  |  |  |  |  |  | x | x | x | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | x | x | x | X |  |  |  |  |  |  |  |  |  |  |
| t | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |

1. **Priority scheduling with preemption (smaller values state higher priority)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | x | x |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  | x | x | x | x | x | X | X |  |  |  |  |
| P2 |  |  | x | x | x | x | x | x | x | x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | x | X | X |  |
| P4 |  |  |  |  |  |  |  |  |  |  | x | x | x | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | x | x | X | X |  |  |  |  |  |  |  |  |  |  | x |
| t | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |

**Q5) (15pt)** Assume that there is a memory system with 4 frames and the memory is initially empty. **FIFO** and **LRU** page replacement algorithms are applied for the page requests given below. Show the contents of the frames at each step on a table, state the steps that the page faults occur, and if a page replacement occurs state the page which is replaced out. Compare the given page replacement algorithms using your findings.

“0 1 2 3 2 4 1 3 5 6 1 3 2 7 4 8”

**X means page fault occured**

i) FIFO

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 2 | 4 | 1 | 3 | 5 | 6 | 1 | 3 | 2 | 7 | 4 | 8 |
| Frame 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 8 |
| Frame 1 |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 5 | 5 | 5 | 5 | 2 | 2 | 2 | 2 |
| Frame 2 |  |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 6 | 6 | 6 | 6 | 7 | 7 | 7 |
| Frame 3 |  |  |  | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 1 | 1 | 1 | 1 | 4 | 4 |
| Page Fault | x | x | x | x |  | x |  |  | x | X | X | X | x | x | x | x |

ii) LRU

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 2 | 4 | 1 | 3 | 5 | 6 | 1 | 3 | 2 | 7 | 4 | 8 |
| Frame 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 4 | 6 | 6 | 6 | 6 | 7 | 7 | 7 |
| Frame 1 |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 4 | 4 |
| Frame 2 |  |  | 2 | 2 | 2 | 2 | 2 | 2 | 5 | 5 | 5 | 5 | 2 | 2 | 2 | 2 |
| Frame 3 |  |  |  | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 8 |
| Page Fault | x | x | x | x |  | x |  |  | x | x |  |  | x | x | x | X |

**For the given sequence of pages the LRU alghoritm seems to be more efficent as it have caused 2 less page faults compared to FIFO. LRU causes less page faults because of the assumption that pages which have been accessed most recently are more likely to be accessed again.**

**Q6) (15pt)** In a computer system, there are free memory partitions as 100 KB, 400 KB, 200 KB and 500KB (in given order). Where would the following memory allocation algorithms place memory requests of 200 KB, 396 KB, 100KB and 290KB, arriving in the given order?

i) First-fit

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Slots | 100 | 400 | 200 | 500 |
| processes | 100 | 200 |  | 396 |

No space is left for 290 in this case

ii) Best-fit

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Slots | 100 | 400 | 200 | 500 |
| processes | 100 | 396 | 200 | 290 |

iii) Next-fit

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Slots | 100 | 400 | 200 | 500 |
| processes | 100 | 200 |  | 396 |

No space is left for 290 in this case

iv) Worst-fit

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Slots | 100 | 400 | 200 | 500 |
| processes |  | 396 | 100 | 200  290 |

v) Please explain which allocation method utilizes the memory most efficiently and why?

**In this case the best fit alghoritm works the best because it leaves the biggest amount of contagious memory on the memory. We also see that the worst fit alghoritm utilises the slot with 500 mb of memory the best out of the 4 methods but when compared to best fit it leaves memory in three parts as 100-4-100-10 where the best fit alghoritm leaves us with 4-310 which is more prefferable. For the other alghoritms we see that we actually run out of space which indicates that those ones are less successfull.**