## Technical University of Denmark

Written Examination, December 11, 2024

Course: Operating Systems

Course No.: 02159

Aids allowed: All aids

Exam duration: 4 hours

Weighting: According to their respective points (100 points in total)

**Notes:**

- All questions contribute toward the final grade. The maximum number of points awarded by each correctly answered question is listed next to the question.
- For short questions, the following apply: (i) a brief answer is expected; (ii) you are also expected to briefly explain your answer in approximately 2-3 lines; (iii) a correct answer with a correct explanation yields 4 points in total.
- For long questions, the following apply: (i) there might be more than one correct answer; (ii) you are expected to briefly explain your answer in approximately half a page; (iii) a complete and correct answer yields 12 points; (iv) if you make additional assumptions, remember to briefly explain them.

### A. Short Questions

**Question A.1:** What is the key advantage of monolithic operating systems compared to microkernel operating systems? Explain briefly your answer. (**4 points**)

**Question A.2:** You are developing an application that is composed of multiple collaborative processes. You wish to implement the following functionality: if a resource is currently unavailable, the process should go to sleep until it receives a wakeup signal from another the process. Which method you would use to avoid a race condition? Explain briefly your answer. (**4 points**)

**Question A.3:** What is the primary role of the Memory Management Unit (MMU)? Explain briefly your answer. (**4 points**)

**Question A.4:** Explain the challenges associated with static relocation and Why it is necessary to transition to dynamic relocation. Discuss the advantages of dynamic relocation in overcoming these challenges. (**4 points**)

**Question A.5:** What will the code in Figure 1 most likely print? Explain briefly your answer. You can assume that all system call invocations are successful. (**4 points**)

**Question A.6:** What will the code in Figure 2 print? Explain briefly your answer. You can assume that all system call invocations are successful. (**4 points**)

**Question A.7:** Which of the 4 answers will the code in Figure 3 most likely print? Explain briefly your answer. You can assume that all system call invocations are successful.

A) 10000

B) 1

C) An integer less than 10000

D) An integer greater than 10000

If you believe that the correct answer is different from A, provide a modification to the code to make it print 10000.(**4 points**)

```
int accounts[2];
pthread_mutex_t lock[2];
pthread_t tid1, tid2;

struct Transact {
  int from;
  int to;
  int amount;
};

struct Transact t1 = {.from = 0, .to = 1, .amount = 10};
struct Transact t2 = {.from = 1, .to = 0, .amount = 20};

void *transfer(void *in) {
  struct Transact trs = *(struct Transact*)in;

  pthread_mutex_lock(&lock[trs.from]);
  sleep(1);
  pthread_mutex_lock(&lock[trs.to]);

  accounts[trs.from]-=trs.amount;
  accounts[trs.to]+=trs.amount;

  pthread_mutex_unlock(&lock[trs.to]);
  pthread_mutex_unlock(&lock[trs.from]);
  pthread_exit(0);
}

int main(void) {
  pthread_mutex_init(&lock[0], NULL);
  pthread_mutex_init(&lock[1], NULL);
  accounts[0] = 100;
  accounts[1] = 100;

  pthread_create(&tid1, NULL, transfer, (void*)&t1);
  pthread_create(&tid2, NULL, transfer, (void*)&t2);
  pthread_join(tid1, NULL);
  pthread_join(tid2, NULL);
  printf("%d\n", accounts[0]);
}
```

Figure 1: Code of Question A.5.

```
int counter = 1;

void child(void) {
    counter++;
    exit(0);
}

int main(void)
{
 if (fork() == 0) {
   child();
 }
 waitpid(-1, NULL, 0);

 printf("%d\n", counter);
 return 0;
}
```

Figure 2: Code of Question A.6.

```
#define THREAD_COUNT 10000
int A = 0;
pthread_t thread_id[THREAD_COUNT];

void* count(void *input) {
  A++;
  pthread_exit(NULL);
}

int main(void) {
  int i;
  for (i=0;i<THREAD_COUNT;i++)
    pthread_create(&thread_id[i], NULL, count, NULL);
  for (i=0;i<THREAD_COUNT;i++)
    pthread_join(thread_id[i], NULL);
  printf("%d\n", A);
  return 0;
}
```

Figure 3: Code of Question A.7.

```
int main(void){
  int a = 0;
  fork();
  printf("%d\n", a);
  fork();
  ++a;
  printf("%d\n", a);
  fork();
  waitpid(-1, NULL, 0);
  exit(0);
}
```

Figure 4: Code of Question A.9.

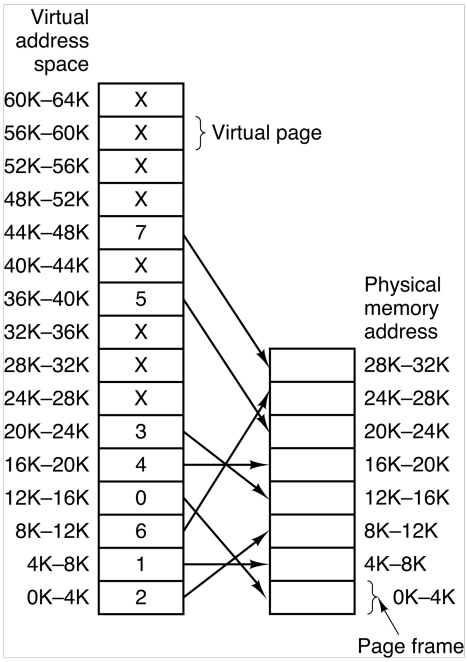


Figure 5: Data for Question A.11.

**Question A.8:** Describe a bug or error that would trigger a watchdog timer reboot? Explain briefly your answer. (**4 points**)

**Question A.9:** What will the code in Figure 4 print , and what is the exact role of waitpid in this example? Explain briefly your answer. You can assume that all system call invocations are successful. (**2 points**)
How many distinct copies of the parent process's memory are created during the execution of this code? Explain your reasoning (**2 points**)

**Question A.10:** Which is the key advantage of developing device drivers using interrupt-driven I/O? Explain briefly your answer. (**4 points**)

**Question A.11:** Assuming the current state of the memory is as shown in Figure 5, name a virtual address that will generate a page fault if accessed? Explain briefly your answer. (**2 points**)
Given that the OS uses FIFO as page replacement algorithm and its current list is [ 6,1,3,5,2,4,0,7 ] with the rightmost element representing the tail of the list. What is the physical address corresponding to the virtual address after the appropriate page is loaded? Explain briefly your answer. (**2 points**)

**Question A.12:** Name a system operation that is not possible without clock interrupts. Explain briefly your answer. (**4 points**)

**Question A.13:** Nodes in a distributed system typically use protocols that have been standardised by international standardisation bodies to communicate with each other. Are international standards also necessary when nodes in a multicomputer system communicate with each other? Explain briefly your answer. (**4 points**)

**Question A.14:** Which is the key disadvantage of the NFU (Not Frequently Used) page replacement algorithm? Explain briefly your answer. (**4 points**)

**Question A.15:** Consider preemptive and non-preemptive scheduling. Identify two real-life applications/senarios where:
(i)Preemptive scheduling is better suited than non-preemptive scheduling.
(ii)Non-preemptive scheduling is more appropriate than preemptive scheduling.
For each example, explain briefly why you chose the particular scheduling technique and how it benefits the application. (**4 points**)

**Question A.16:** The DMA chip can transfer data from and to the memory without using the CPU. Name a scenario that it is not efficient to use DMA for memory transfer? Explain briefly your answer. (**4 points**)

## B. Long Questions

### Question B.1 (12 points)

(i) Describe, in your own words, how the scheduling algorithm of Linux prioritises the execution of processes that cannot tolerate latency.

(ii) Let's assume that you are designing a server for the OS Challenge. Arriving requests have 99% probability to have priority 1 and 1% probability to have priority 200. How would you design parallelism and prioritisation? You shall assume that anything not explicitly specified in this sub-question is as described on the OS Challenge specification document.

### Question B.2 (12 points)

(i) Summarise the advantages and disadvantages of processes and threads.

(ii) Give one example of a real-world application where you would choose to implement it with multiple threads and one example that you would choose to implement it with multiple processes. Motivate your design decision. Do not use the example applications that we used in class.

### Question B.3 (12 points)

(i) Explain in your own words when using a spinlock (*i.e.*, a busy-waiting mutex) in a multiprocessor system can improve the performance.

(ii) You are using a multiprocessor system. The operating system implements hybrid mutexes. These mutexes operate as follows. If a thread requests to acquire a locked mutex, the thread first continuously polls the mutex (spins) for a period of time, $T$. After the time $T$ passes, the thread yields (context switch) and retries when it gets rescheduled. The parameter $T$ is configurable, taking values in $\mu$s in the range $[0, 65535]$. Setting $T = 0$ disables spinning.

After long-term statistical analysis, you know that the time a thread needs to wait for a locked mutex to be released follows the histogram provided in Figure 6. Moreover, a context switch takes 1000 $\mu$s.

Calculate the optimum value for the configuration parameter $T$, which minimises the overhead (*i.e.* the sum of time the CPU wastes spinning and switching). For simplicity, you can consider that when the thread gets rescheduled after the first context switch, it finds the mutex unlocked.

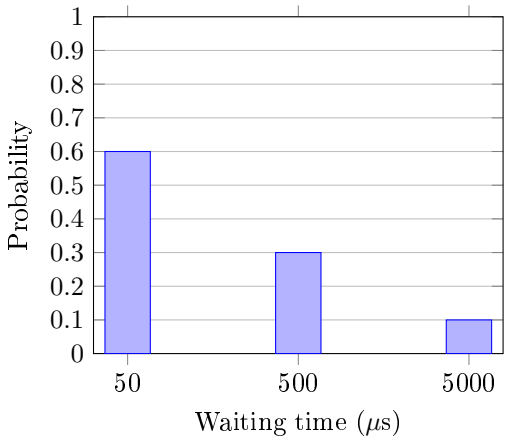


Figure 6: Data for Question B.3.