Chalmers University of Technology and Gothenburg University

Operating Systems
EDA093, DIT 401

Exam 2025-01-07

Date, Time: Tuesday 2025/01/07, 08.30-12.30

Course Responsible:
Vincenzo Gulisano (031 772 61 47)

Auxiliary material: You may have with you
- An English-Swedish, Swedish-English dictionary.
- No other books, notes, calculators, etc.

Grade-scale ("Betygsgränser"):
CTH: 3:a 30-39 p, 4:a 40-49 p, 5:a 50-60 p
GU: Godkänd 30-49p, Väl godkänd 50-60 p

Exam review ("Granskningstid"):
Will be announced after the exam.

Instructions

- Do not forget to write your personal number, if you are a GU or CTH student and at which program ("linje").
- Start answering each assignment on a new page; number the pages and use only one side of each sheet of paper.
- Write in a **clear manner** and **motivate** (explain, justify) your answers. If it is not clear what is written, your answer will be considered wrong. If it is not explained/justified, even a correct answer will get **significantly** lower (possibly zero) marking.
- Please make an effort to have nice calligraphy!
- If you make **any assumptions** in answering any item, do not forget to clearly state what you assume.
- Answer questions in English, if possible. If you have a large difficulty with that and you think that your grade can be affected, feel free to write in Swedish.

Good luck !!!!

1. (6 p) Would it make sense for an attacker to exploit a buffer overflow in the user rather than in the kernel code? If you believe the answer is yes, provide an example.

   [**Answer:** Yes, depending on the attacker's goals and the system's security measures, it can make sense for an attacker to exploit a buffer overflow in the user code rather than in the kernel code. The kernel code is often well-protected. However, user code may have fewer protections and still provide valuable opportunities for attackers.

   Consider a buffer overflow in a file management application running in the user space. By exploiting the vulnerability, an attacker injects malicious code that deletes specific files belonging to the user, such as important documents or configuration files.]

2. (6 p) True/False Questions – Kernel Threads and Kernel Mode (provide a short explanation about your True/False answer):

   (a) Kernel threads always execute in kernel mode.
   [**Answer:** Answer: False. Kernel threads can execute in either kernel mode or user mode depending on their specific task.]

   (b) Switching between kernel threads of the same process is faster than switching between processes.
   [**Answer:** Answer: True. Context switching between kernel threads of the same process is faster because they share the same address space.]

   (c) User-level threads are managed by the operating system kernel, while kernel threads are managed by the user-level threading library.
   [**Answer:** Answer: False. Kernel threads are managed by the operating system kernel, while user-level threads are managed by the user-level threading library.]

   (d) When a process with multiple kernel threads makes a blocking system call, only the thread making the call is blocked.
   [**Answer:** Answer: True. In a system supporting kernel threads, other threads of the process can continue executing.]

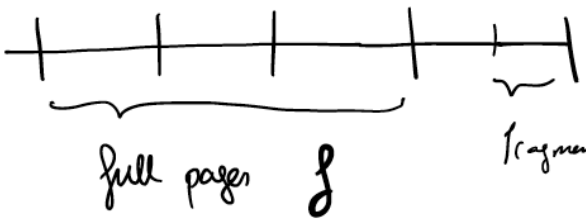
   (e) Kernel threads are always associated with a user process.
   [**Answer:** Answer: False. Kernel threads can exist independently of user processes (e.g., threads for handling I/O or other kernel-level tasks).]

   (f) Kernel mode is always required for switching between kernel threads.
   [**Answer:** Answer: True. Switching between kernel threads involves accessing kernel-level resources and requires kernel mode.]

3. (6 p) In virtual memory, if the page size 2^n increases, does the internal fragmentation necessarily increase/stay the same, or can it also decrease? Motivate your answer.

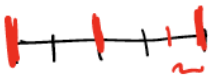
   [**Answer:** only stay the same or increase. Some visual explanation follows:]

Size 2^m

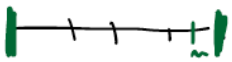


case 1 $(f+1) \% 2 = 0$ (the number of used frames is even)

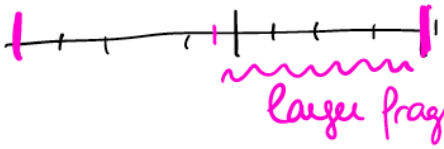
$M = m + 1$

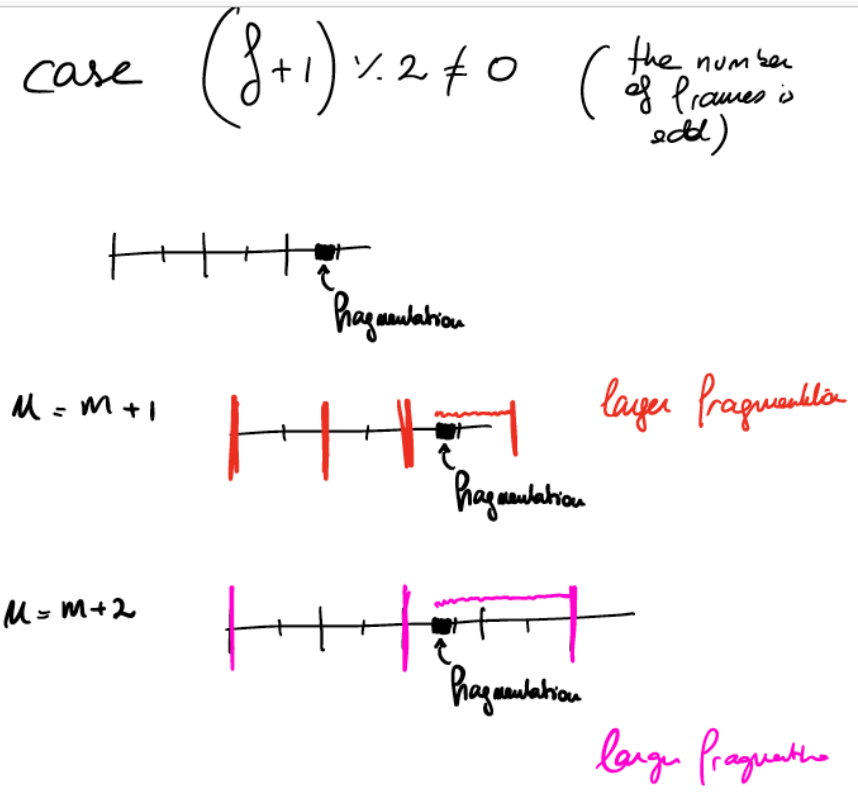


$M = m + 2$



$M = m + 3$





4. (6 p) The manual of `vfork()` states:

   > "The `vfork()` function has the same effect as `fork(2)`, except that the behavior is undefined if the process created by `vfork()` either modifies any data other than a variable of type `pid_t` used to store the return value from `vfork()`, or returns from the function in which `vfork()` was called, or calls any other function before successfully calling `_exit(2)` or one of the `exec(3)` family of functions."

   Why?

   [**Answer:** The `vfork()` function is designed to create a new process in a highly efficient manner by sharing the address space of the parent process with the child process, rather than copying it as `fork()` does. This optimization cannot guarantee a certain behavior for the parent if the child "messes" with the parent's memory. That is because, unlike `fork()`, the child process does not receive a separate copy of the parent's address space. Instead, the parent and child processes share the same memory. If the child modifies any data in this shared memory, it can lead to unpredictable behavior or corrupt the parent's state. In other words, the purpose of `vfork()` is to allow the child to immediately call `exec()` or `exit()`. Any other action, such as returning from the function or calling other functions, risks modifying the shared memory or stack, leading to undefined behavior.]

5. (6 p) Does Direct Memory Access reduce the number of interrupts in a computer system? Explain why or why not in detail.

   [**Answer:** Yes, Direct Memory Access (DMA) reduces the number of interrupts in a computer system. In a system without DMA, the CPU is responsible

for transferring data between memory and I/O devices. This often requires multiple interrupts. With DMA, the CPU delegates the data transfer task to the DMA controller. The DMA controller handles the entire transfer between memory and the I/O device without CPU intervention. Instead of multiple interrupts for each small transfer, the DMA controller generates a single interrupt to notify the CPU when the entire transfer is complete. This significantly reduces the number of interrupts, particularly for large data transfers, and minimizes CPU overhead allowing the CPU to focus on other tasks.]

6. (6 p) Write C code (simplified) that creates N threads and distributes them to M processes in a round-robin fashion (i.e., if $N = 10$ and $M = 3$, threads $1, 4, 7, \ldots$ belong to process 1, threads $2, 5, 8, \ldots$ to process 2, and so on). All threads belonging to each process should be able to exchange information through a shared variable.

   [**Answer:** Answer:]

```c
#include ...

void *thread_function(void *arg) {...}

int shared_var;

int main(int argc, char *argv[]) {
    if (argc != 3) {
        print error...
    }

    int N = atoi(argv[1]); // Number of threads
    int M = atoi(argv[2]); // Number of processes

    pid_t pids[M];

    for (int i = 0; i < M; i++) {
        pids[i] = fork();
        if (pids[i] == 0) {
            // Child process
            pthread_t threads[N / M + 1];
            int thread_count = 0;

            // Create threads for this process
            for (int j = 0; j < N; j++) {
                if (j % M == i) {
                    pthread_create(&threads[thread_count],
                        NULL, thread_function, shared_data)
                        ;
                    thread_count++;
                }
            }

            // Wait for threads to finish
            for (int j = 0; j < thread_count; j++) {
                pthread_join(threads[j], NULL);
            }

            exit(EXIT_SUCCESS);
        }
    }

```

```
41      // Parent process waits for all child processes
42      for (int i = 0; i < M; i++) {
43          waitpid(pids[i], NULL, 0);
44      }
45
46      return;
47  }
```

7. (6 p) Upon which events can the OS scheduler take scheduling decisions if non-preemptive scheduling is to be enforced?

   [**Answer:** In a non-preemptive scheduling system, the operating system scheduler can make scheduling decisions only at specific points where the currently running process voluntarily releases the CPU. These events include: process creation, process termination, when a process blocks, when it issues a wait request, for instance upon an I/O request.]

8. (6 p) Show how the dining philosophers problem can be solved by breaking the circular wait condition.

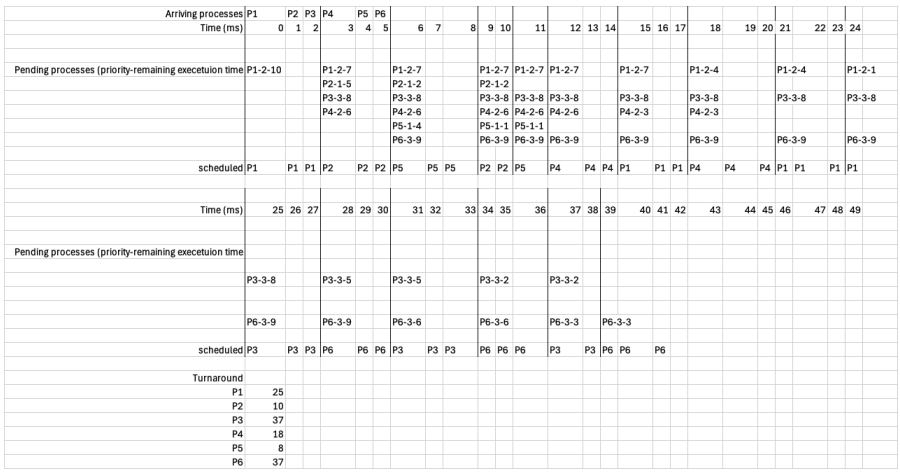
   [**Answer:** Check slide 20 in Lecture 6.]

9. (6 p) Consider the following six processes, each with a priority (lower number indicates higher priority), a CPU burst time (in milliseconds), and an arrival time (in milliseconds):

Process	Priority	Burst Time (ms)	Arrival Time (ms)
P1	2	10	0
P2	1	5	1
P3	3	8	2
P4	2	6	3
P5	1	4	5
P6	3	9	6

   Assume the CPU uses a round-robin scheduling algorithm with a time quantum of 3 milliseconds. However, at each quantum, the process with the highest priority (lowest priority number) is selected for execution. Compute the turnaround time for each process.

   [**Answer:** Answer:]



10. (6 p) Consider the following C code:

```c
#include ...

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        return 0;
    } else if (pid > 0) {
        sleep(10);
    } else {
        return 1;
    }

    return 0;
}
```

Does this program lead to a zombie process or an orphan process? Justify your answer (assume that the `sleep(10)` instruction implies the process entering the branch for which `pid==0` holds true has plenty of time to run its code).

[**Answer:** This program creates a **zombie process**. A zombie process occurs when a child terminates but the parent has not yet collected its exit status. When the child process terminates (indicated by 'return 0' in the child block), it remains in the process table as a zombie until its parent explicitly reads its exit status using the 'wait()' or 'waitpid()' system call. The parent process does not call 'wait()' immediately; instead, it sleeps for 10 seconds. During this time, the child process remains in the zombie state, occupying a slot in the process table. It is not an orphan process because the parent process is still running and has not terminated. An orphan process occurs when a child process continues execution after its parent has terminated, and it gets adopted by the 'init' process.]