ECE 434: Intro to Computer Systems – Spring 2025

Adil Hydari

Homework 1

Issue Date: Wednesday 02-19-2025
Due Date: Friday 02-28-2025 at 11:59 pm

Contents

1 Problem 1: Interrupts (3 pts) 2

2 Problem 2: Direct Memory Access (DMA) (6 pts) 3

3 Problem 3: Familiarizing with Linux (6 pts) 4

4 Problem 4: Altering the Simple Instruction Cycle (3 pts) 5

5 Problem 5: DMA Quantitative Problem (6 pts) 6

6 Problem 6: Modes of Data Transfer (9 pts) 8

7 Problem 7: C Programming – Files (10 pts) 9

8 Problem 8: Linux Shell Scripting (10 pts) 18

1 Problem 1: Interrupts (3 pts)

Description:
Describe what a hardware interrupt, a trap, and an exception is, and list and justify as many differences as you can identify between any two of those.

Answer

- **Hardware Interrupt:** A signal from external hardware (e.g., an I/O device) to the CPU indicating that it needs immediate attention. Hardware interrupts are *asynchronous* to program execution. When they occur, the CPU suspends its current instruction flow after finishing the current instruction and jumps to the appropriate interrupt service routine (ISR).
- **Trap:** A software-initiated interrupt. Typically triggered by a special instruction (e.g., a system call or a deliberate "breakpoint"). Traps are *synchronous* events because they occur at a precise point during instruction execution (the point at which the program issues the instruction).
- **Exception:** A synchronous event caused by the CPU detecting an error/exception condition while executing an instruction (e.g., division by zero, invalid memory access, page fault). Exceptions cause control to transfer to an exception handler, which decides how to recover or terminate.

Differences (example pairwise comparisons):

1. Hardware Interrupt vs. Trap:
   - **Source**: Hardware interrupts come from external devices; traps are requested by software.
   - **Timing**: Hardware interrupts are asynchronous; traps are synchronous with the instruction stream.
2. Trap vs. Exception:
   - **Intent**: A trap is generally an intentional software event (e.g., a system call), while an exception is often unintentional.
   - **Handling**: Traps often pass control to well-defined OS routines, whereas exceptions might also raise error signals and can terminate or fix the failing instruction.

---

2 Problem 2: Direct Memory Access (DMA) (6 pts)

Q1 (2 pts)

Question:
Assume that a Network Card has data to transfer with the end goal to save the data in DRAM. Describe in detail the steps that need to be taken by the CPU, the Network Card, the user program, and the OS to coordinate the transfer (type of interrupts, register modifications, mode shifts, etc.).

Answer

1. The **user program** initiates a receive operation (e.g., via `recv()` system call).
2. The **OS kernel** is invoked. In kernel mode, it allocates or identifies a DRAM buffer to store the incoming data.
3. The OS programs the **DMA controller** by writing:
   - The starting (base) address in DRAM.
   - The size (bound) or length of the buffer.
   - Possibly other control bits for the DMA engine.
4. The **Network Card** then takes over the actual transfer via DMA hardware. The CPU is free to context-switch to other tasks.
5. Once the DMA completes, the Network Card raises a **DMA completion interrupt**.
6. The CPU receives the interrupt, switches to **interrupt handler** in kernel mode, and checks the status registers.
7. The OS marks the data buffer as "ready," and then the user process is unblocked or notified that the data has arrived.

Q2 (2 pts)

Question: How does the CPU know when the memory operations are complete?

Answer

The device sends a **hardware interrupt** to signal completion. The interrupt service routine (ISR) in the OS then processes the completion event (e.g., by marking the data buffer valid).

Q3 (2 pts)

Question: The CPU is allowed to execute other programs while the DMA controller is transferring data. Does this process interfere at any point during its life-cycle with the execution of user programs? If so, describe when and how interference(s) are caused.

Answer

Yes, there are two points:

1. **Bus Contention**: The DMA controller uses the system bus to read/write memory. When the bus is busy for DMA, the CPU may experience slight performance delays if it also needs the bus.
2. **Interrupt Handling**: When the DMA completes, the device raises an interrupt, forcing the CPU to switch to kernel mode to service it. This interrupt handling momentarily stalls user programs.

---

3 Problem 3: Familiarizing with Linux (6 pts)

1. **(1 pt) Which instruction gives you your default shell?**
   Answer:

   ```
   echo $SHELL
   ```

2. **(1 pt) Which instruction gives you the current shell?**
   Answer:

   ```
   echo $0
   ```

3. **(1 pt) Which instruction gives you the process ID (PID) of your current shell?**
   Answer:

   ```
   echo $$
   ```

4. **(1 pt) Which instructions do you need to use to switch from your default shell to another shell and back to your default shell?**
   Answer: To switch shells, just type the name of the shell, e.g., `zsh` or `bash`. To go back, type `exit`.

5. **(2 pts) Open two different shells in two different terminals, get information from both using `ls` (with a valid flag), `ps` (with a valid flag), or `htop`, then terminate the older shell from the newer shell. Show a screenshot.**
   Answer (example steps):

(a) Terminal 1: run `echo $$` to see the PID (5678).

(b) Terminal 2: run `ps -a` or `ps -ux` to confirm shell's PID is 5678 in Terminal 1.

(c) In Terminal 2, type `kill 5678`.

(d) Terminal 1 shell is terminated.

Screenshots would be attached here.

---

4 Problem 4: Altering the Simple Instruction Cycle (3 pts)

Description:
We have user space addresses > 100 and kernel space addresses ≤ 100. The `mode` register is 0 in kernel mode; 1 in user mode. Modify instruction fetch–decode–execute to handle memory violations or a timer interrupt. Then explain how the OS handles simultaneous events.

Q1

Question: Which states/phases of handling an instruction (fetch–decode–execute) must be modified? Provide pseudo-code.

Answer

We need to check for:

- **Memory violation**: If the CPU (in user mode) tries to access an address ≤ 100, or an invalid address.
- **Timer interrupt**: If the timer expires.

```
Pseudo-code:

FETCH:
IR <- MEM[PC]
if (violation_detected or timer_interrupt_pending) then
goto HANDLE_INTERRUPT
PC <- PC + 4

DECODE:
decode IR
if (timer_interrupt_pending) then
goto HANDLE_INTERRUPT
```

```
EXECUTE:
execute instruction in IR
if (memory_violation or timer_interrupt_pending) then
goto HANDLE_INTERRUPT

HANDLE_INTERRUPT:
save_current_context()
if memory_violation then
// invoke memory violation
// kill the user process or handle exception
else if timer_interrupt_pending then
// run timer handler
restore_context()
```

Q2

Question: What if the two events occur almost simultaneously?

Answer

In most systems, interrupts have priority levels. For instance, a memory violation might have higher priority than a timer interrupt. The CPU or interrupt controller decides which interrupt to service first based on priority. If the timer interrupt is pending while a memory violation occurs, the memory-violation handler is invoked first. After handling the higher-priority event, the CPU can check if other interrupts are still pending.

5 Problem 5: DMA Quantitative Problem (6 pts)

Part (a) (2 pts)

Question: In the Polled I/O, why did we need to modify the Naïve Receive (RX) to Polled Receive but it would be ok to leave the Transmit (TX) to execute a while loop?

Answer

- **Naïve RX:** Continuously spinning (busy-waiting) for incoming data wastes CPU cycles. Polled receive only checks status when needed, or waits until an interrupt or periodic poll.
- **TX in a while loop:** Transmit simply pushes data out. Once we load the transmit buffer, the hardware will handle it. We only wait (in a loop) for the hardware to

be "ready." Because we are not missing data if we wait to transmit, this is less of a problem performance-wise.

Part (b) (4 pts)

Question: We have:

- Load: 300 clock cycles
- Store: 250 clock cycles
- Interrupt latency (1st interrupt): 2,500 instructions @ 2 cycles each
- Interrupt latency (2nd interrupt): 800 instructions @ 2 cycles each
- Setup for DMA: 4 loads (for base and bound registers)

Give a numerical example of packet size for which PIO ¡ DMA. Then assume we can transmit more than one byte at a time using a 64-bit register. Which packet sizes favor PIO over DMA?

Answer (Sample Computation)

- **DMA overhead:**

$$\text{DMA setup} = 4 \times (300 \text{ cycles per load}) = 1200 \text{ cycles}$$

$$\text{Interrupt overhead} = (2500+800) \text{ instructions} \times 2 \text{ cycles each} = 3300 \times 2 = 6600 \text{ cycles}$$

$$\text{Total DMA overhead} = 1200 + 6600 = 7800 \text{ cycles}$$

- **PIO overhead (per byte)**: Each byte store = 250 cycles. Let N = number of bytes.

$$\text{PIO cost} = 250 \times N$$

If $250N < 7800$, then PIO is faster. That implies:

$$N < \frac{7800}{250} \approx 31.2$$

So, for example, a 30-byte packet has:

$$\text{PIO cost} = 30 \times 250 = 7500 < 7800$$

Hence, 30 bytes is an example where PIO is faster.

- **64-bit memory-mapped register**: - 64 bits = 8 bytes at a time. - Then the cost per 8-byte store is still ≈ 250 cycles (one store instruction). - For N bytes, we need $\frac{N}{8}$ stores, so

$$\text{PIO cost} = \left(\frac{N}{8}\right) \times 250$$

Set that < 7800:

$$\frac{N}{8} \times 250 < 7800 \quad \Rightarrow \quad N < \frac{7800 \times 8}{250} = 249.6$$

So for packets up to around 249 bytes, PIO is still faster; above that, DMA overhead pays off.

---

6 Problem 6: Modes of Data Transfer (9 pts)

Q1 (2 pts)

Question: Why did we need to modify the Naïve Receive (RX) to Polled Receive but leave the Transmit task (TX) to execute a while loop?

Answer

It closely mirrors Problem 5(a):

- **RX (Receive)**: Spinning naively can waste CPU cycles checking for new data. Polled receive is more efficient because we only check when necessary or when scheduled.
- **TX (Transmit)**: It is less harmful to wait in a while loop for the transmitter to be ready before sending the next chunk, because there is no risk of "missing" inbound data.

Q2 (2 pts)

Question: In Programmed I/O (PIO), what are the benefits of using Memory-Mapped Registers?

Answer

- The CPU can use the same instructions (`load`/`store`) to communicate with devices rather than special I/O instructions.
- Simplifies CPU design and can make device driver code simpler, because device registers appear in the address space.
- Compiler and OS can treat device registers similarly to normal memory, which can ease portability.

Q3 (5 pts)

Question: Polled vs. DMA Data Communication

1. (2 pts) In a Polled system, when the Status Registers change, a trap to the OS occurs. How often or under what circumstances does this trap happen? Describe the process and hardware components involved when a trap takes place.

2. (3 pts) Why are the DMA interrupts more expensive than the trap in the Polled system? Is the trap in the Polled RX actually an interrupt? If so, why is there such a big difference in clock cycles compared to DMA interrupts?

Answer

1. Frequency of Trap in Polled RX:

   - Typically, the CPU executes a loop (in the driver or OS) checking a device's status register. Whenever it detects that the status bits changed (e.g., new data arrived), it issues a *software trap* (or system call) into kernel mode to process that data.
   - Alternatively, some architectures arrange it so that changes in a memory-mapped status register cause an exception, but most often it is simply the polling code that triggers a trap-like request to handle data.

2. DMA Interrupts vs. Polled Traps:

   - **DMA has two expensive interrupts**: one to start/prepare the transfer, another to signal completion. Both involve full interrupt-service overhead: saving CPU state, switching to kernel mode, possibly flushing pipelines or TLB, etc.
   - **Polled trap overhead** is often lower because the CPU "knows" it is about to transition to kernel mode (the polling code explicitly checks and then calls the service routine). This avoids some overhead compared to an asynchronous hardware interrupt, which may force a pipeline flush, context save, and interrupt vector lookup.
   - So yes, the polled trap is effectively a form of interrupt, but is usually simpler or partially handled in software, incurring fewer cycles than a hardware-driven, asynchronous interrupt with full context switching overhead.

---

7 Problem 7: C Programming – Files (10 pts)

Tasks:

1. Complete a program `merge_files` that merges two input files into a third file. The default output is `myfile.out`.

2. Show sample runs, including `strace` output.

Answer

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  // Check for correct usage
  if (argc < 3) {
    printf("Usage: %s file_in_1 file_in_2 [file_out (default: myfile.out)]\n",
           argv[0]);
    return 1;
  }

  // Attempt to open first two input files
  FILE *fp1 = fopen(argv[1], "r");
  FILE *fp2 = fopen(argv[2], "r");
  if (!fp1) {
    printf("%s: No such file or directory\n", argv[1]);
    return 1;
  }
  if (!fp2) {
    printf("%s: No such file or directory\n", argv[2]);
    fclose(fp1);
    return 1;
  }

  // Determine output file name
  char *outFileName = (argc == 4) ? argv[3] : "myfile.out";

  FILE *fp3 = fopen(outFileName, "w");
  if (!fp3) {
    printf("Could not open or create output file %s\n", outFileName);
    fclose(fp1);
    fclose(fp2);
    return 1;
  }

  // Copy contents from first file
```

```
    int ch;
    while ((ch = fgetc(fp1)) != EOF) {
        fputc(ch, fp3);
    }

    // Copy contents from second file
    while ((ch = fgetc(fp2)) != EOF) {
        fputc(ch, fp3);
    }

    // Close files
    fclose(fp1);
    fclose(fp2);
    fclose(fp3);

    printf("Merged %s and %s into %s\n", argv[1], argv[2], outFileName);

    return 0;
}
```

Makefile

```
CC = gcc
CFLAGS = -Wall -fsanitize=address

all:
$(CC) $(CFLAGS) -o merge_files merge_files.c

clean:
rm -f merge_files
```

Sample Runs

```
$ echo "Data for file_in_1 This is OS 434 Sp25," > A1
$ echo "data for file_in_2 but also OS 579 Sp25!" > A2

$ ./merge_files A1 A2
Merged A1 and A2 into myfile.out
$ cat myfile.out
Data for file_in_1 This is OS 434 Sp25,
data for file_in_2 but also OS 579 Sp25!
```

```
$ ./merge_files A1 A2 A3
Merged A1 and A2 into A3
$ cat A3
Data for file_in_1 This is OS 434 Sp25,
data for file_in_2 but also OS 579 Sp25!
```

`strace` Output

```
execve("./merge_files", ["./merge_files"], 0x7ffd227a0cc0 /* 44 vars */) = 0
brk(NULL)                               = 0x5ab066c8d000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=259939, ...}) = 0
mmap(NULL, 259939, PROT_READ, MAP_PRIVATE, 3, 0) = 0x708c4e429000
close(3)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4270
openat(AT_FDCWD, "/usr/lib/libasan.so.8", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=12004152, ...}) = 0
mmap(NULL, 7558504, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x708c4dc00000
mmap(0x708c4dc32000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
mmap(0x708c4ddba000, 262144, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1ba00
mmap(0x708c4ddfa000, 32768, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
mmap(0x708c4de02000, 5453160, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0Pz\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 840, 64)
fstat(3, {st_mode=S_IFREG|0755, st_size=2022936, ...}) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 840, 64)
mmap(NULL, 2047160, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x708c4da0c000
mmap(0x708c4da30000, 1503232, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
mmap(0x708c4db9f000, 339968, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19300
mmap(0x708c4dbf2000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
mmap(0x708c4dbf8000, 31928, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/../lib/glibc-hwcaps/x86-64-v4/libstdc++.so.6", O_RDONLY|O_CL
newfstatat(AT_FDCWD, "/usr/lib/../lib/glibc-hwcaps/x86-64-v4/", 0x7ffee2380810, 0) = -1
openat(AT_FDCWD, "/usr/lib/../lib/glibc-hwcaps/x86-64-v3/libstdc++.so.6", O_RDONLY|O_CL
newfstatat(AT_FDCWD, "/usr/lib/../lib/glibc-hwcaps/x86-64-v3/", 0x7ffee2380810, 0) = -1
openat(AT_FDCWD, "/usr/lib/../lib/glibc-hwcaps/x86-64-v2/libstdc++.so.6", O_RDONLY|O_CL
newfstatat(AT_FDCWD, "/usr/lib/../lib/glibc-hwcaps/x86-64-v2/", 0x7ffee2380810, 0) = -1
```

```
openat(AT_FDCWD, "/usr/lib/../lib/libstdc++.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=25830928, ...}) = 0
mmap(NULL, 2928768, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x708c4d600000
mmap(0x708c4d694000, 1638400, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
mmap(0x708c4d824000, 614400, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22400
mmap(0x708c4d8ba000, 57344, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
mmap(0x708c4d8c8000, 12416, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/../lib/libm.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=854360, ...}) = 0
mmap(NULL, 856392, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x708c4e355000
mmap(0x708c4e364000, 544768, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
mmap(0x708c4e3e9000, 245760, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x94000
mmap(0x708c4e425000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/../lib/libgcc_s.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=1006416, ...}) = 0
mmap(NULL, 205288, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x708c4d9d9000
mmap(0x708c4d9dd000, 167936, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
mmap(0x708c4da06000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2d000)
mmap(0x708c4da0a000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3
close(3)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e3530
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e350
arch_prctl(ARCH_SET_FS, 0x708c4e3507c0) = 0
set_tid_address(0x708c4e350a90)         = 11013
set_robust_list(0x708c4e350aa0, 24)     = 0
rseq(0x708c4e350680, 0x20, 0, 0x53053053) = 0
mprotect(0x708c4dbf2000, 16384, PROT_READ) = 0
mprotect(0x708c4da0a000, 4096, PROT_READ) = 0
mprotect(0x708c4e425000, 4096, PROT_READ) = 0
mprotect(0x708c4d8ba000, 53248, PROT_READ) = 0
mprotect(0x708c4ddfa000, 20480, PROT_READ) = 0
mprotect(0x5ab035b86000, 4096, PROT_READ) = 0
mprotect(0x708c4e4a6000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=16384*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x708c4e429000, 259939)          = 0
readlinkat(AT_FDCWD, "/proc/self/exe", "/home/adilh/latex-notes-hw/comp_"..., 4096) = 5
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4680
```

```
openat(AT_FDCWD, "/proc/self/cmdline", O_RDONLY) = 3
read(3, "./merge_files\0", 4096)        = 14
read(3, "", 4082)                       = 0
close(3)                                = 0
munmap(0x708c4e468000, 4096)            = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4680
openat(AT_FDCWD, "/proc/self/environ", O_RDONLY) = 3
read(3, "LC_ADDRESS=en_US.UTF-8\0MOTD_SHOW"..., 4096) = 1133
read(3, "", 2963)                       = 0
close(3)                                = 0
mmap(NULL, 65536, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e458
mmap(NULL, 3727360, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4d2
mmap(NULL, 2097152, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4d0
munmap(0x708c4d100000, 1048576)         = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4570
mmap(NULL, 2097152, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4ce
munmap(0x708c4cf00000, 1048576)         = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4560
mmap(NULL, 2097152, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4cc
munmap(0x708c4cd00000, 1048576)         = 0
mmap(NULL, 2097152, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4ca
munmap(0x708c4cb00000, 1048576)         = 0
mmap(NULL, 2097152, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4c8
munmap(0x708c4c900000, 1048576)         = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4550
prlimit64(0, RLIMIT_CORE, NULL, {rlim_cur=RLIM64_INFINITY, rlim_max=RLIM64_INFINITY}) =
prlimit64(0, RLIMIT_CORE, {rlim_cur=0, rlim_max=RLIM64_INFINITY}, NULL) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4540
openat(AT_FDCWD, "/proc/self/maps", O_RDONLY) = 3
read(3, "5ab035b83000-5ab035b84000 r--p 0"..., 4096) = 4091
read(3, "708c4", 5)                     = 5
close(3)                                = 0
munmap(0x708c4e454000, 4096)            = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4530
openat(AT_FDCWD, "/proc/self/maps", O_RDONLY) = 3
read(3, "5ab035b83000-5ab035b84000 r--p 0"..., 8192) = 4091
read(3, "708c4e49b000-708c4e4a6000 r--p 0"..., 4101) = 523
read(3, "", 3578)                       = 0
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4520
openat(AT_FDCWD, "/proc/self/maps", O_RDONLY) = 3
read(3, "5ab035b83000-5ab035b84000 r--p 0"..., 4096) = 4091
```

```
read(3, "708c4", 5)                     = 5
close(3)                                = 0
munmap(0x708c4e452000, 4096)            = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4510
openat(AT_FDCWD, "/proc/self/maps", O_RDONLY) = 3
read(3, "5ab035b83000-5ab035b84000 r--p 0"..., 8192) = 4091
read(3, "708c4e49b000-708c4e4a6000 r--p 0"..., 4101) = 523
read(3, "", 3578)                       = 0
close(3)                                = 0
munmap(0x708c4e451000, 8192)            = 0
mmap(0x7fff7000, 268435456, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS|M
madvise(0x7fff7000, 268435456, MADV_NOHUGEPAGE) = 0
madvise(0x7fff7000, 268435456, MADV_DONTDUMP) = 0
mmap(0x2008fff7000, 15392894357504, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANO
madvise(0x2008fff7000, 15392894357504, MADV_NOHUGEPAGE) = 0
madvise(0x2008fff7000, 15392894357504, MADV_DONTDUMP) = 0
mmap(0x8fff7000, 2199023255552, PROT_NONE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS|MAP_NORE
sigaltstack(NULL, {ss_sp=NULL, ss_flags=SS_DISABLE, ss_size=0}) = 0
mmap(NULL, 57344, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e445
sigaltstack({ss_sp=0x708c4e445000, ss_flags=0, ss_size=54016}, NULL) = 0
rt_sigaction(SIGSEGV, {sa_handler=0x708c4dd4b890, sa_mask=[], sa_flags=SA_RESTORER|SA_O
rt_sigaction(SIGBUS, {sa_handler=0x708c4dd4b890, sa_mask=[], sa_flags=SA_RESTORER|SA_ON
rt_sigaction(SIGFPE, {sa_handler=0x708c4dd4b890, sa_mask=[], sa_flags=SA_RESTORER|SA_ON
mmap(0x500000000000, 4398046519296, PROT_NONE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS|MAP_
mmap(0x540000000000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -
mmap(NULL, 8388608, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_NORESERVE, -1, 0) = 0x708c
mmap(NULL, 57344, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e437
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4360
getpid()                                = 11013
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=16384*1024, rlim_max=RLIM64_INFINITY}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4350
openat(AT_FDCWD, "/proc/self/maps", O_RDONLY) = 3
read(3, "7fff7000-8fff7000 rw-p 00000000 "..., 4096) = 4085
read(3, "708c4e46d00", 11)              = 11
close(3)                                = 0
munmap(0x708c4e435000, 4096)            = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4340
openat(AT_FDCWD, "/proc/self/maps", O_RDONLY) = 3
read(3, "7fff7000-8fff7000 rw-p 00000000 "..., 8192) = 4085
read(3, "708c4e46d000-708c4e46f000 r-xp 0"..., 4107) = 809
read(3, "", 3298)                       = 0
close(3)                                = 0
```

```
munmap(0x708c4e453000, 8192)            = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4540
openat(AT_FDCWD, "/proc/self/maps", O_RDONLY) = 3
read(3, "7fff7000-8fff7000 rw-p 00000000 "..., 4096) = 4047
read(3, "708c4e46b000-708c4e46d000 r--p 0"..., 49) = 49
close(3)                                = 0
munmap(0x708c4e454000, 4096)            = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4530
openat(AT_FDCWD, "/proc/self/maps", O_RDONLY) = 3
read(3, "7fff7000-8fff7000 rw-p 00000000 "..., 8192) = 4085
read(3, "708c4e46d000-708c4e46f000 r-xp 0"..., 4107) = 809
read(3, "", 3298)                       = 0
close(3)                                = 0
munmap(0x708c4e453000, 8192)            = 0
mmap(0x10005c269000, 2093056, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS
madvise(0x10005c269000, 2093056, MADV_NOHUGEPAGE) = 0
madvise(0x10005c269000, 2093056, MADV_DONTDUMP) = 0
mmap(NULL, 11571200, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4b
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4540
sigaltstack(NULL, {ss_sp=0x708c4e445000, ss_flags=0, ss_size=54016}) = 0
mmap(NULL, 1703936, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4b3
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4530
munmap(0x708c4e453000, 4096)            = 0
mmap(NULL, 2097152, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4b0
munmap(0x708c4b100000, 1048576)         = 0
mmap(NULL, 2097152, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4ae
munmap(0x708c4af00000, 1048576)         = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4530
mmap(NULL, 65536, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e340
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4320
clock_gettime(CLOCK_MONOTONIC, {tv_sec=1961, tv_nsec=978255191}) = 0
mmap(0x506000000000, 65536, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
mmap(0x506e00000000, 65536, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
mmap(NULL, 1048576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4d8
mmap(NULL, 8388608, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_NORESERVE, -1,
clock_gettime(CLOCK_MONOTONIC, {tv_sec=1961, tv_nsec=978850719}) = 0
mmap(0x503000000000, 65536, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
mmap(0x503e00000000, 65536, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
futex(0x708c4d8c86bc, FUTEX_WAKE_PRIVATE, 2147483647) = 0
mmap(0x531000000000, 131072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
mmap(0x531e00000000, 65536, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4e4310
```

```
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
mmap(0x519000000000, 65536, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
mmap(0x519e00000000, 65536, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
write(1, "Usage: ./merge_files file_in_1 f"..., 73Usage: ./merge_files file_in_1 file_i
) = 73
gettid()                                = 11013
gettid()                                = 11013
prctl(PR_GET_DUMPABLE)                  = 1 (SUID_DUMP_USER)
getpid()                                = 11013
mmap(NULL, 2101248, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4b1
mprotect(0x708c4b156000, 4096, PROT_NONE) = 0
rt_sigprocmask(SIG_BLOCK, ~[ILL ABRT BUS FPE SEGV XCPU XFSZ], [], 8) = 0
clone(child_stack=0x708c4b356ff0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_UNTRACED) =
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
getpid()                                = 11013
prctl(PR_SET_PTRACER, 11014)            = 0
futex(0x7ffee23813b8, FUTEX_WAKE_PRIVATE, 1) = 1
sched_yield()                           = 0
sched_yield()                           = 0
sched_yield()                           = 0
sched_yield()                           = 0
sched_yield()                           = 0
sched_yield()                           = 0
sched_yield()                           = 0
sched_yield()                           = 0
wait4(11014, NULL, __WALL, NULL)        = 11014
munmap(0x708c4b156000, 2101248)         = 0
getpid()                                = 11013
write(2, "==11013==LeakSanitizer has encou"..., 54==11013==LeakSanitizer has encountere
) = 54
mmap(NULL, 65536, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x708c4d262
getpid()                                = 11013
write(2, "==11013==HINT: For debugging, tr"..., 102==11013==HINT: For debugging, try se
) = 102
getpid()                                = 11013
write(2, "==11013==HINT: LeakSanitizer doe"..., 75==11013==HINT: LeakSanitizer does not
) = 75
exit_group(1)                           = ?
+++ exited with 1 +++
```

---

8 Problem 8: Linux Shell Scripting (10 pts)

Description:
Create 8 text files, each with a random number of lines. Write a shell script that:

- Takes a number of these files as arguments.
- Counts lines in each, adds them cumulatively, prints results to screen and a file, and shows total files processed.

Answer (`line_counter.sh`)

```
#!/bin/bash

if [ $# -lt 1 ]; then
  echo "Usage: $0 file1 [file2 ...]"
  exit 1
fi

# Output file (append or overwrite as needed)
output_file="line_count_results.txt"
>"$output_file"

total_lines=0
count_files=0

for f in "$@"; do
  if [ -f "$f" ]; then
    # Count lines
    lines_in_file=$(wc -l <"$f")
    total_lines=$((total_lines + lines_in_file))
    count_files=$((count_files + 1))
    echo "File: $f has $lines_in_file lines." | tee -a "$output_file"
  else
    echo "Warning: $f is not a valid file." | tee -a "$output_file"
  fi
done

echo "----------------------------------------" | tee -a "$output_file"
echo "Number of files processed: $count_files" | tee -a "$output_file"
echo "Total number of lines: $total_lines" | tee -a "$output_file"
```

How to Run

1) Create 8 new .txt files (e.g., f1.txt, f2.txt, etc.) and fill them with random line

```
2) Make script executable:  chmod +x line_counter.sh
3) Run:  ./line_counter.sh f1.txt f2.txt ...
4) Output is shown on screen and stored in line_count_results.txt
```

Output

```
~/latex-notes-hw/comp_systems/hw1 comp_systems
- ./line_counter.sh A1 A2
File: A1 has 1 lines.
File: A2 has 1 lines.
---------------------------------
Number of files processed: 2
Total number of lines: 2
```