Politechnika Wrocławska

Programowanie Systemowe

LABORATORIUM

Temat: **Pamięć współdzielona i semafory**

Hanna Kieszek – 283797

17.12.2025

1. Cel ćwiczenia

Celem ćwiczenia było zmodyfikowanie pliku z poprzednich zajęć tak aby do komunikacji między workerem a serverem została wykorzystana pamięć współdzielona oraz semafory.

2. Wykorzystane narzędzia

1. Ubuntu
2. Terminal

3. Przebieg laboratorium

3.1. Zmiana danych wejściowych w Makefile

Plik Makefile posłuży do skompilowania plików które zostały utworzone podczas tego laboratorium. Poniżej znajduje się zmodyfikowany Makefile:

```
 GNU nano 7.2                                Makefile
OUTDIR=build
SRCDIR=projectv2
CC=gcc
TARGET=$(OUTDIR)/plik
SRC=$(wildcard $(SRCDIR)/*.c)
OBJ=$(patsubst $(SRCDIR)/%.c, $(OUTDIR)/%.o, $(SRC))

all: $(TARGET)

clean:
        rm -rf $(OUTTDIR)

$(TARGET): $(OBJ)
        $(CC) $^ -o $@

$(OUTDIR)/%.o: $(SRCDIR)/%.c | $(OUTDIR)
        $(CC) -c $< -o $@

$(OUTDIR):
        mkdir $@
```

Obraz 1: Wygląd pliku Makefile

Pliki z poprzednich zajęć zostały zmodyfikowane w taki sposób aby była możliwa dwukierunkowa komunikacja między workerem a serverem poprzez wprowadzenie pamięci współdzielonej. Zasada działania i zakres obowiązków servera i workera nie uległy zmianie. Oprócz tego zostały wprowadzone semafory aby sterować dostępem do pamięci współdzielonej.

3.2. Modyfikacja pliku main.c

Na początku pliku main.c zostały zadeklarowane stałe oraz nagłówki. Następnie rozpoczyna się funkcja main() która przyjmuje argumenty. Jeżeli zostanie wpisana niewłaściwa liczba argumentów program zwróci błąd co zostało zdefiniowane w pierwszej instrukcji warunkowej. Następnie rozpoczyna się druga w której program wykonuje co innego w zależności od wpisanego argumentu.

Pierwsza jej część dotyczy argumentu “server”. Na początku została zdefiniowana pamięć współdzielona za pomocą funkcji: shm_open() - utworzenie obiektu pamięci współdzielonej, ftruncate() - ustawienie rozmiaru pamięci, mmap() - mapowanie pamięci. Następnie zostały utworzone dwa semafory sem_worker i sem_server za pomocą funkcji sem_open(), które mają początkową wartość równą 0. W kolejnej linijce jest wywoływana funkcja server() która jako argumenty przyjmuje wskaźnik do pamięci współdzielonej oraz dwa semafory. Kiedy ta funkcja się zakończy, zostają zamknięte semafory oraz pamięć współdzielona.

Druga część instrukcji warunkowej dotyczy argumentu “worker”. Zostaje otwarty istniejący obiekt pamięci współdzielonej a następnie pamięć ta zostaje zmapowana do przestrzeni adresowej procesu workera. Następnie zostają otwarte semafory które również posiadają wartości początkowe równe 0. W kolejnej linijce zostaje wywołana funkcja worker() z takimi samymi argumentami jak server(). Na koniec jak funkcja się zakończy, semafory oraz pamięć współdzielona zostają zamknięte.

Na następnej stronie znajduje się kod pliku main.c

GNU nano 7.2 projectv3/main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <semaphore.h>
#include "server.h"
#include "worker.h"

#define SHM_NAME "/shm_lab"
#define SEM_SERVER "/sem_server"
#define SEM_WORKER "/sem_worker"
#define SHM_SIZE 100

int main(int argc, char *argv[]) {
                int shm_fd;
                char *shm_ptr;

                sem_t *sem_server;
                sem_t *sem_worker;

                if (argc != 2)
                {
                        fprintf(stderr, "Niewlasciwa liczba argumentow\n");
                        return 1;
                }

                if (strcmp(argv[1], "server") == 0)
                {

                        shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
                        ftruncate(shm_fd, SHM_SIZE);

                        shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);

                        sem_server = sem_open(SEM_SERVER, O_CREAT, 0666, 0);
                        sem_worker = sem_open(SEM_WORKER, O_CREAT, 0666, 0);

                        server(shm_ptr, sem_server, sem_worker);

                        munmap(shm_ptr, SHM_SIZE);
                        close(shm_fd);

                        sem_close(sem_server);
                        sem_close(sem_worker);

                        sem_unlink(SEM_SERVER);
                        sem_unlink(SEM_WORKER);
                        shm_unlink(SHM_NAME);
        }

        else if (strcmp(argv[1], "worker") == 0) {

                shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);

                shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);

                sem_server = sem_open(SEM_SERVER, 0);
                sem_worker = sem_open(SEM_WORKER, 0);

                worker(shm_ptr, sem_server, sem_worker);

                munmap(shm_ptr, SHM_SIZE);
                close(shm_fd);

                sem_close(sem_server);
                sem_close(sem_worker);
        }

        return 0;
}
```

3.3. Modyfikacja pliku server.c

Plik server.c rozpoczyna się od zadeklarowania nagłówków a następnie znajduje się definicja funkcji server(). W tej definicji znajduje się pętla while(1) w której znajdują się instrukcje wykonywane w pętli dopóki nie zostanie wpisane słowo kończące pętlę czyli “exit”. Na początku funkcja odczytuje input użytkownika i zapisuje go do pamięci współdzielonej. Następnie zostaje zwiększona wartość semafora za pomocą funkcji sem_post(sem_worker). Dzięki temu worker zostaje odblokowany i poinformowany o nowych danych w pamięci współdzielonej. Następnie znajduje się instrukcja warunkowa dotycząca słowa kończącego “exit”. Na końcu znajduje się funkcja sem_wait(sem_server) dzięki której server oczekuje na sygnał od workera, że zakończył on przetwarzać dane, aby mógł wznowić swoje działanie. Jeżeli otrzyma taki sygnał, wyświetli zmodyfikowany tekst z pamięci współdzielonej.

```
 GNU nano 7.2                    projectv3/server.c
#include <stdio.h>
#include <string.h>
#include <semaphore.h>
#include "server.h"

void server(char *shm_ptr, sem_t *sem_server, sem_t *sem_worker) {
        while(1) {

                printf("in: ");
                scanf("%99s", shm_ptr);
                sem_post(sem_worker);


                if(strcmp(shm_ptr, "exit") == 0)
                {
                        break;
                }

                sem_wait(sem_server);
                printf("out: %s\n", shm_ptr);
        }
}
```

```
 GNU nano 7.2                    projectv3/server.h
#ifndef SERVER_H
#define SERVER_H

#include <semaphore.h>

void server(char *shm_ptr, sem_t *sem_server, sem_t *sem_worker);

#endif
```

3.4. Modyfikacja pliku worker.c

Na samym początku zostały zadeklarowane nagłówki po których następuje zdefiniowanie funkcji worker(), która również składa się z pętli while() która wykonuje się w nieskończoność dopóki nie zostanie wpisane słowo kończące “exit”. W pierwszej linijce pętli znajduje się funkcja sem_wait(sem_worker), która powoduje wstrzymanie działania workera dopóki nie dostanie sygnału od servera. Kiedy dostanie taki sygnał, ma dostęp do pamięci współdzielonej w której znajduje się tekst wprowadzony przez użytkownika. Worker edytuje dane pamięci współdzielonej shm_ptr[0]=’X’; i daje sygnał serverowi, że dane są gotowe i może wznowić swoje działanie.

```
 GNU nano 7.2                    projectv3/worker.c
#include <stdio.h>
#include <string.h>
#include <semaphore.h>
#include "worker.h"

void worker(char *shm_ptr, sem_t *sem_server, sem_t *sem_worker) {
        while(1) {
                sem_wait(sem_worker);

                if (strcmp(shm_ptr, "exit") == 0)
                {
                        sem_post(sem_server);
                        break;
                }
                shm_ptr[0]='X';

                sem_post(sem_server);
        }
}
```

```
 GNU nano 7.2                    projectv3/worker.h
#ifndef WORKER_H
#define WORKER_H

#include <semaphore.h>

void worker(char *shm_ptr, sem_t *sem_server, sem_t *sem_worker);

#endif
```

3.5. Działanie programu

Kompilacja programu za pomocą Makefile:

```
ubuntu@Ubuntu:~$ make
mkdir build2
gcc -c projectv3/main.c -o build2/main.o
gcc -c projectv3/server.c -o build2/server.o
gcc -c projectv3/worker.c -o build2/worker.o
gcc build2/main.o build2/server.o build2/worker.o -o build2/plik
```

Działanie programu: