## Idea programu:

Program składa się z dwóch części : server i worker. Oba programy są uruchamiane w dwóch osobnych terminalach.

Komunikacja między nimi odbywa się przez:

pamięć współdzieloną (shared memory) – wspólne miejsce na dane; semafory – żeby kontrolować, kto i kiedy może korzystać z pamięci.

Działanie:

1. Server wpisuje tekst do pamięci
2. Worker odczytuje tekst z pamięci
3. Worker zmienia pierwsza litera na X
4. Server odczytuje zmieniony tekst i wyświetla go
5. Wpisanie exit kończy program

## main.c:

rozpoznaje, czy program działa jako server czy jako worker, tworzy pamięć współdzieloną i semafory dla servera, otwiera istniejące zasoby dla workera, uruchamia odpowiednią funkcję: server_loop() lub worker_loop()

#define SHM_NAME "/shm_lab" - nazwa pamięci współdzielonej w systemie

#define SEM_SERVER "/sem_server"; #define SEM_WORKER "/sem_worker" - nazwy semaforów: jeden kontroluje server, drugi worker

#define SHM_SIZE 100 - rozmiar pamięci współdzielonej – 100 znaków

int main(int argc, char *argv[]) - przyjmuje argument: server lub worker

**część server:**

if (strcmp(argv[1], "server") == 0) - sprawdzamy, czy program działa jako server

shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666) - tworzymy pamięć współdzieloną:

- O_CREAT – jeśli nie istnieje, utwórz
- O_RDWR – odczyt i zapis
- 0666 – prawa dostępu

ftruncate(shm_fd, SHM_SIZE) - ustawiamy rozmiar pamięci na 100 bajtów

shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0) - mapujemy pamięć do programu

sem_server = sem_open(SEM_SERVER, O_CREAT, 0666, 1) - tworzymy semafor dla server: wartość początkowa = 1 -> server może działać

sem_worker = sem_open(SEM_WORKER, O_CREAT, 0666, 0) - tworzymy semafor dla worker: wartość = 0 -> worker na początku czeka

pętla servera:

sem_wait(sem_server) - server czeka na swoją kolej

scanf("%s", shm_ptr) - użytkownik wpisuje tekst, który trafia do pamięci współdzielonej

sem_post(sem_worker) - server informuje workera, że dane są gotowe do odczytu

sem_wait(sem_server) - server czeka, aż worker skończy modyfikację tekstu

printf("SERVER: od worker: %s\n", shm_ptr) - server wyświetla zmodyfikowany tekst

munmap(shm_ptr, SHM_SIZE) - odłączenie pamięci\

sem_close(sem_server); sem_close(sem_worker) - zamykanie semaforów

sem_unlink(SEM_SERVER); sem_unlink(SEM_WORKER); shm_unlink(SHM_NAME) - server usuwa pamięć i semafory z systemu

**część worker:**

else if (strcmp(argv[1], "worker") == 0) - program działa jako worker

shm_fd = shm_open(SHM_NAME, O_RDWR, 0666) - worker otwiera istniejącą pamięć i podłącza ją do programu - 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 otwiera istniejące semafory

pętla workera:

sem_wait(sem_worker) - worker czeka, aż server wyśle dane

shm_ptr[0] = 'X' - worker zmienia pierwszą literę tekstu na "X”

sem_post(sem_server) - worker informuje servera, że dane są gotowe

munmap(shm_ptr, SHM_SIZE); close(shm_fd); sem_close(sem_server); sem_close(sem_worker) - zamykanie przez worker

### server.c:

server.c odpowiada za:

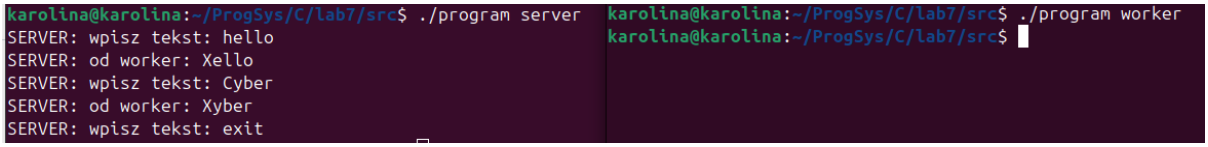
- pobieranie tekstu od użytkownika
- zapisywanie go do pamięci
- odbieranie zmodyfikowanego tekstu od worker
- wyświetlanie wyniku

### worker.c:

worker.c:

- czeka na dane od servera
- czyta tekst z pamięci
- zmienia pierwszą literę na X
- informuje servera, że gotowe

Wynik działania:




program został uruchomiony w dwóch terminalach: w pierwszym jako server, w drugim jako worker

server wysyła tekst do pamięci współdzielonej, worker odczytuje tekst i zmienia pierwszą literę na “X”, a następnie zapisuje zmodyfikowany tekst z powrotem do pamięci, server odczytuje wynik i wyświetla go użytkownikowi

po wpisaniu “exit” program kończy działanie