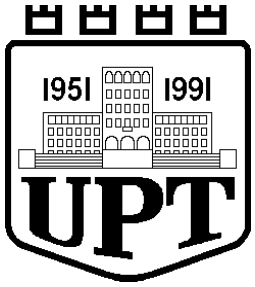

UNIVERSITETI POLITEKNIK I TIRANËS
FAKULTETI I TEKNOLOGJISË DHE INFORMACIONIT
DEPARTAMENTI I INXHINIERISË INFORMATIKE

Punë Laboratori nr. 4

Tema: Përdorimi I semaforëve në C

Lënda: Programim në Sisteme Të Shpërndara

Grupi: III-B

Punoi:
Piro Gjikdhima

Pranoi:
MSc.Megi Tartari

Ushtrimi 1

Shkruaj një program në gjuhën C për të simuluar problemin e prodhuesit-konsumator duke përdorur semaforë.

Problemi i **prodhuesit-konsumator** është një paradigmë e zakonshme për proceset që bashkëpunojnë. Një proces **prodhues** prodhon informacion që konsumohet nga një proces **konsumator**. Një zgjidhje për problemin e prodhuesit-konsumator përdor **memorie të përbashkët**. Për të lejuar që proceset prodhues dhe konsumator të ekzekutohen njëkohësisht, duhet të ekzistojë një **buffer (tampon)** ku prodhuesi mund të vendosë artikuj dhe konsumatori t'i marrë ato. Ky buffer ndodhet në një rajon memorieje që është i përbashkët mes prodhuesit dhe konsumatorit.Një **prodhues** mund të prodhojë një artikull ndërkohë që konsumatori po konsumon një tjetër. Prodhuesi dhe konsumatori duhet të **sinkronizohen**, në mënyrë që konsumatori të mos përpiqet të konsumojë një artikull që ende nuk është prodhuar.

KODI

producer_consumer.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define BUFFER_SIZE 5
#define NUM_PRODUCERS 3
#define NUM_CONSUMERS 2
#define PRODUCE_COUNT 5
#define CONSUME_COUNT ((NUM_PRODUCERS * PRODUCE_COUNT) / NUM_CONSUMERS)

int buffer[BUFFER_SIZE];
int in = 0, out = 0;

sem_t empty;
sem_t full;
pthread_mutex_t mutex;

void *producer(void *arg)
{
    int id = *((int *)arg);
    int item;
    for (int i = 0; i < PRODUCE_COUNT; i++)
    {
        item = rand() % 100;
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
```

```
        buffer[in] = item;
        printf("Producer %d prodhoi: %d\n", id, item);
        in = (in + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);
        sem_post(&full);

        sleep(1);
    }
    return NULL;
}

void *consumer(void *arg)
{
    int id = *((int *)arg);
    int item;
    for (int i = 0; i < CONSUME_COUNT; i++)
    {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);

        item = buffer[out];
        printf("Consumer %d konsumoi: %d\n", id, item);
        out = (out + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);
        sem_post(&empty);

        sleep(1);
    }
    return NULL;
}

int main()
{
    pthread_t producers[NUM_PRODUCERS];
    pthread_t consumers[NUM_CONSUMERS];
    int prod_ids[NUM_PRODUCERS];
    int cons_ids[NUM_CONSUMERS];

    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    pthread_mutex_init(&mutex, NULL);

    for (int i = 0; i < NUM_PRODUCERS; i++)
    {
        prod_ids[i] = i + 1;
```

```
        pthread_create(&producers[i], NULL, producer, &prod_ids[i]);
    }

    for (int i = 0; i < NUM_CONSUMERS; i++)
    {
        cons_ids[i] = i + 1;
        pthread_create(&consumers[i], NULL, consumer, &cons_ids[i]);
    }

    for (int i = 0; i < NUM_PRODUCERS; i++)
    {
        pthread_join(producers[i], NULL);
    }

    for (int i = 0; i < NUM_CONSUMERS; i++)
    {
        pthread_join(consumers[i], NULL);
    }

    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Rezultate

1. Deklarimi i Matricave dhe Variablave**:**

- **buffer[BUFFER_SIZE]**: Një buffer i ndarë për prodhuesit dhe konsumatorët.
- **in dhe out**: Indekset për të monitoruar pozicionet ku elementet futen dhe konsumohen.
- **empty dhe full**: Semaforë për të kontrolluar gjendjen e boshatisjes dhe mbushjes në buffer.
- **mutex**: Mutex për të siguruar akses në buffer nga prodhuesit dhe konsumatorët.

2. Funksioni **producer():**

- Secili prodhues krijon një produkt të rastësishëm dhe e vendos atë në buffer.
- Përdoret **sem_wait(&empty)** për të kontrolluar nëse ka hapësirë në buffer dhe **sem_post(&full)** për të njoftuar konsumatorët kur ka një produkt të disponueshëm.
- Përdoret **pthread_mutex_lock(&mutex)** dhe **pthread_mutex_unlock(&mutex)** për të siguruar akses të sigurt në buffer.

3. Funksioni **consumer():**

- Secili konsumator merr një produkt nga buffer dhe e konsumon.

- Përdoret **sem_wait(&full)** për të kontrolluar nëse ka produkte për konsumim dhe **sem_post(&empty)** për të njoftuar prodhuesit që ka hapësirë në buffer.
- Si prodhuesit, konsumatorët përdorin **pthread_mutex_lock(&mutex)** dhe **pthread_mutex_unlock(&mutex)** për të parandaluar konkurencën për buffer.

4. Funksioni **main():**

- Inicializohet **semafori** dhe **mutex**.
- Krijohen dhe nisin **prodhuesit dhe konsumatorët** për të punuar në mënyrë të pavarur .
- Pas përfundimit të procesit, thirren **pthread_join()** për të pritur përfundimin e thread-eve dhe pastaj shkatërrohen semaforët dhe mutex-i.

```
Producer 1 prodhoi: 41
Producer 2 prodhoi: 41
Producer 3 prodhoi: 41
Consumer 1 konsumoi: 41
Consumer 2 konsumoi: 41
Consumer 1 konsumoi: 41
Producer 1 prodhoi: 67
Consumer 2 konsumoi: 67
Producer 2 prodhoi: 67
Producer 3 prodhoi: 67
Producer 2 prodhoi: 34
Producer 3 prodhoi: 34
Consumer 2 konsumoi: 67
Producer 1 prodhoi: 34
Consumer 1 konsumoi: 67
Consumer 2 konsumoi: 34
Producer 2 prodhoi: 0
Producer 3 prodhoi: 0
Consumer 1 konsumoi: 34
Producer 1 prodhoi: 0
Consumer 1 konsumoi: 34
Producer 1 prodhoi: 69
Consumer 2 konsumoi: 0
Producer 3 prodhoi: 69
Producer 2 prodhoi: 69
Consumer 1 konsumoi: 0
Consumer 2 konsumoi: 0
Consumer 2 konsumoi: 69
Consumer 1 konsumoi: 69
```

Ushtrimi 2

Shkruaj një program në gjuhën C për të simuluar algoritmin e Bankierit për qëllimin e shmangies së bllokimit (deadlock).

Në një mjedis me shumë programe (multiprogramim), disa procese mund të konkurrojnë për një numër të kufizuar burimesh. Një proces kërkon burime; nëse burimet nuk janë të disponueshme në atë moment, procesi kalon në një gjendje pritjeje. Ndonjëherë, një proces që është në pritje nuk është më në gjendje të ndryshojë gjendje, sepse burimet që ka kërkuar mbahen nga procese të tjera që gjithashtu janë në pritje. Kjo situatë quhet **bllokim**. **Shmangia e bllokimit** është një nga teknikat për trajtimin e bllokimeve. Kjo qasje kërkon që sistemi operativ të ketë paraprakisht informacione shtesë në lidhje me cilat burime një proces do të kërkojë dhe do të përdorë gjatë jetës së tij. Me këtë njohuri shtesë, sistemi mund të vendosë për çdo kërkesë nëse procesi duhet të vazhdojë apo të presë.

Për të vendosur nëse kërkesa aktuale mund të plotësohet apo duhet të shtyhet, sistemi duhet të marrë parasysh:

- burimet që janë aktualisht të disponueshme,
- burimet që janë aktualisht të alokuara për secilin proces,
- si dhe kërkesat dhe lirimet e ardhshme të secilit proces.

Algoritmi i Bankierit është një algoritëm për shmangien e bllokimeve që aplikohet në një sistem me disa instance (kopje) të secilit lloj burimi.

KODI

bankers_algorithm.c

```c
#include <stdio.h>
#include <stdbool.h>

#define MAX_PROCESSES 10
#define MAX_RESOURCES 10

int processes, resources;
int allocation[MAX_PROCESSES][MAX_RESOURCES];
int max[MAX_PROCESSES][MAX_RESOURCES];
int need[MAX_PROCESSES][MAX_RESOURCES];
int available[MAX_RESOURCES];

void calculateNeed();
bool isSafe();

int main()
```

```
{
    printf("Shkruaj numrin e proceseve: ");
    scanf("%d", &processes);
    printf("Shkruaj numrin e burimeve: ");
    scanf("%d", &resources);

    printf("Shkruaj matricën e alokimit:\n");
    for (int i = 0; i < processes; i++)
    {
        for (int j = 0; j < resources; j++)
        {
            scanf("%d", &allocation[i][j]);
        }
    }

    printf("Shkruaj matricën maksimale:\n");
    for (int i = 0; i < processes; i++)
    {
        for (int j = 0; j < resources; j++)
        {
            scanf("%d", &max[i][j]);
        }
    }

    printf("Shkruaj burimet e disponueshme:\n");
    for (int i = 0; i < resources; i++)
    {
        scanf("%d", &available[i]);
    }

    calculateNeed();

    if (isSafe())
    {
        printf("Sistemi është në një gjendje të sigurt.\n");
    }
    else
    {
        printf("Sistemi nuk është në një gjendje të sigurt.\n");
    }
    return 0;
}

bool isSafe()
{
    int work[MAX_RESOURCES];
    bool finish[MAX_PROCESSES] = {false};
    int safeSequence[MAX_PROCESSES];
```

```
    int index = 0;

    for (int i = 0; i < resources; i++)
    {
        work[i] = available[i];
    }

    int count = 0;
    while (count < processes)
    {
        bool found = false;

        for (int p = 0; p < processes; p++)
        {
            if (!finish[p])
            {
                bool canProceed = true;
                for (int r = 0; r < resources; r++)
                {
                    if (need[p][r] > work[r])
                    {
                        canProceed = false;
                        break;
                    }
                }

                if (canProceed)
                {
                    for (int r = 0; r < resources; r++)
                    {
                        work[r] += allocation[p][r];
                    }
                    finish[p] = true;
                    safeSequence[index++] = p;
                    found = true;
                    count++;
                }
            }
        }

        if (!found)
        {
            return false;
        }
    }

    printf("Rendi i sigurt i proceseve është: ");
    for (int i = 0; i < processes; i++)
```

```
    {
        printf("%d ", safeSequence[i]);
    }
    printf("\n");

    return true;
}

void calculateNeed()
{
    for (int i = 0; i < processes; i++)
    {
        for (int j = 0; j < resources; j++)
        {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
}
```

Rezultate

Deklarimet:

Matricat:

- *allocation*: sa burime janë alokuar për secilin proces.
- *max*: sa burime mund të kërkojë secili proces maksimalisht.
- *need*: sa burime i duhen proceseve për të përfunduar.
- *available*: sa burime janë të disponueshme.

Funksioni **calculateNeed():**

- Ky funksion llogarit matricen **need** duke zbritur matricen e **alokimit** nga matrica e **maksimaleve**. Kjo tregon sa burime i duhen një procesi për të përfunduar.

Funksioni **isSafe():**

- Ky funksion kontrollon nëse sistemi është në një gjendje të sigurt. Përdor një algoritëm të thjeshtë për të gjetur një rend te sigurt (safe sequence) të proceseve që mund të përfundojnë pa shkaktuar bllokim (deadlock):
    - Fillimisht, përpiqet të gjejë një proces që mund të përfundojë duke krahasuar nevojat e tij me burimet e disponueshme.
    - Nëse një proces mund të përfundojë, ai e bën atë dhe çliron burimet.
    - Ky proces vazhdon derisa të gjitha proceset të përfundojnë, duke e bërë sistemin të sigurt.
- Në fund, printohet **renditja e sigurt** e proceseve që mund të ekzekutohen pa probleme.

Funksioni **main()** është pika e hyrjes ku:

- Përdoruesi fut numrin e proceseve, numrin e burimeve, matricat e **alokimit** dhe **maksimaleve**, dhe burimet e **disponueshme**.
- Funksioni **calculateNeed()** llogarit nevojat e proceseve.
- Funksioni **isSafe()** kontrollon nëse sistemi është në një gjendje të sigurt dhe printon përfundimin përkatës.

Algoritmi i Bankierit përdor një qasje **prediktive** për të **parashikuar mundësitë e bllokimit** (deadlock).Përdor **matricat e alokimit dhe maksimale** për të llogaritur nevojat e proceseve dhe **kontrollon nëse mund të përfundojnë** pa bllokuar njëri-tjetrin.Nëse ekziston një **rend i sigurt** (safe sequence) i proceseve, sistemi është **në një gjendje të sigurt**.Algoritmi është përdorur për **menaxhimin e burimeve** në një sistem shumëprocesor dhe shmang bllokimet.

```
Shkruaj numrin e proceseve: 3
Shkruaj numrin e burimeve: 3
Shkruaj matricën e alokimit:
0 1 0
2 0 0
3 0 2
Shkruaj matricën maksimale:
7 5 3
3 2 2
8 0 2
Shkruaj burimet e disponueshme:
3 4 2
Rendi i sigurt i proceseve është: 1 2 0
Sistemi është në një gjendje të sigurt.
```

```
Shkruaj numrin e proceseve: 3
Shkruaj numrin e burimeve: 3
Shkruaj matricën e alokimit:
0 1 0
2 1 1
3 1 2
Shkruaj matricën maksimale:
7 5 3
3 2 2
9 1 2
Shkruaj burimet e disponueshme:
3 3 2
Sistemi nuk është në një gjendje të sigurt.
```