SİSTEM PROGRAMLAMA

Odev 5- IPC olarak Semaforlar ve Paylaşılmış Bellek Kullanımı

GİRİŞ

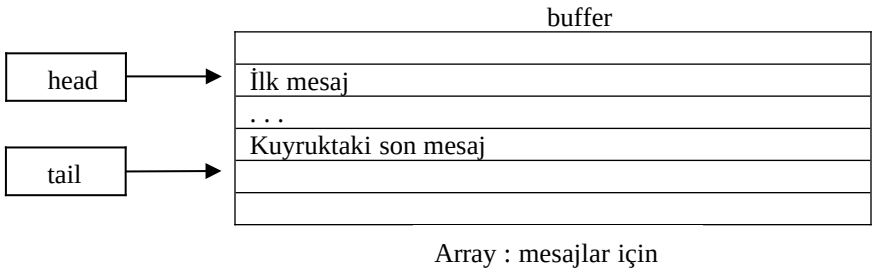
Bu ödevin amacı IPC mekanizması olarak Paylaşılmış Belleğin, semaforlarla nasıl kullanıldığını göstermektir. Paylaşılmış bellek bir çok prosesin aynı anda ortak sanal bir belleği kullanarak iletişimine izin verir. Fakat genellikle paylaşılmış bellek kullanımı, proseslerin hareketlerini koordine etmelerini gerektirir. Diğer türlü, prosesler ortak kaynakları gelişigüzel kullanırsa kritik bölüm (critical sectin) yada yarışma durumu (race condition) gibi problemler gelişebilir. Yani paylaşılmış bellek genelde semaforlarla birlikte kullanılır ki bu sayede prosesler hareketlerini uygun olarak düzenlerler.

Bu ödevde, semaforlarla paylaşılmış bellek bir klasik üretici-tüketici görevi için kullanılacaktır. Bu amaçla, **üç process** yaratılacak ve aşağıdaki gibi devam edecektir. Başlangıçta **parent process** başlar ve aşağıdakileri yapar :

[1] Paylaşılmış belleği yarat, bağla ve sıfırla (başlangıçta belleği sıfırlamak iyi fikir).
[2] İki semafor yarat ve ilklendir.
[3] İki child proses ile producer/consumer programlarını çağır: **üretici** process ve **tüketici** process.
[4] üretici ve tüketici processlerin işlerini bitirmesini bekle, sonra paylaşılmış belleği ve semaforu sistemden sil.

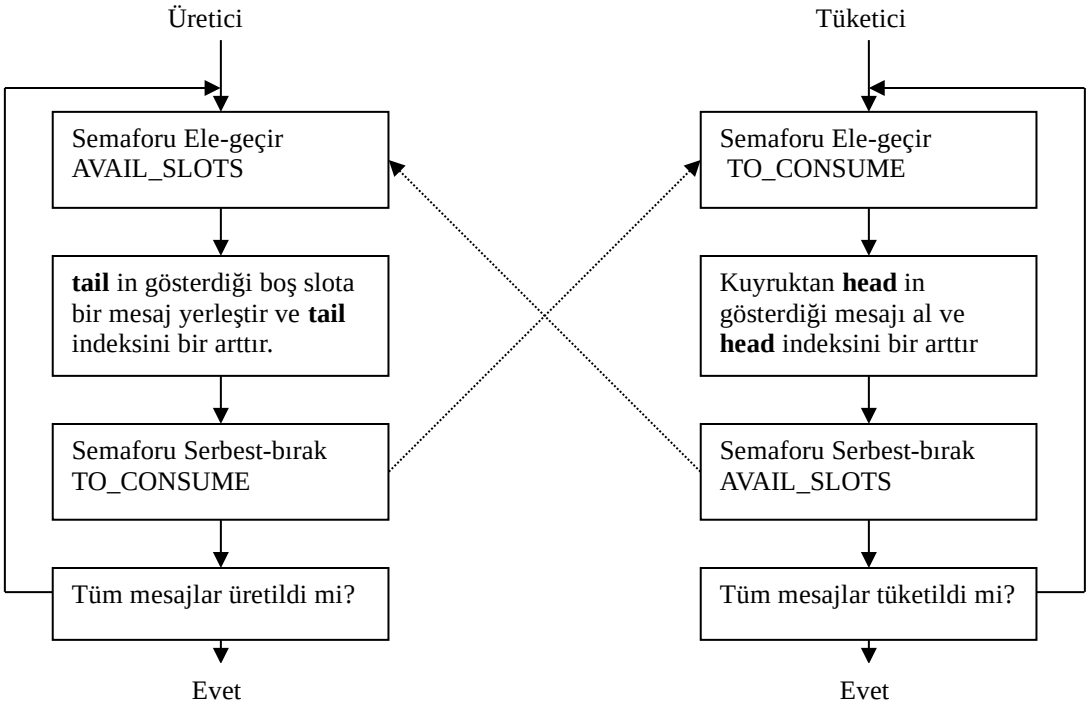
Üretici prosesin görevi, sonlu sayıda mesajları tüketici proses paylaşılmış bellek üzerinden ulaştırmaktır. **Tüketici prosesin** görevi bu mesajları ortak bellekten almak ve ekranda göstermektir. Üreticinin üreteceği mesaj sayısı **komut satırı parametresi** olarak parent prosese oradanda child proseslere geçecektir.

Paylaşılmış Bellek 3 alandan (**buffer, head, tail**) oluşan bir veri yapısı olarak organize edilmiştir (Şekil-1). İlk alan **buffer** üreticinin ürettiği mesajları tutacaktır. Mantıksal olarak buffer, mesaj kuyruğudur. Diğer iki alan **head** ve **tail** ise kuyruğun başını(head) ve sonunu (tail) göstermeleri için kullanılacaktır.



Şekil. 1. Paylaşılmış belleğin örnek veri-yapısı (kuyruk 6 mesajı barındırması için 6 slottan oluşuyor)

Üretici ve tüketicinin ortak belleğe hareketlerini koordine etmeleri için, iki semafor kullanılmıştır. Bir semafor, **AVAIL_SLOTS** (available slots = boş slotlar) mesaj kuyruğundaki **boş slotların sayısını** temsil etmektedir. Üretici ancak bu semafor değeri sıfırdan büyükse, o zaman ürettiği mesajı ortak belleğe atabilir. İkinci semafor, **TO_CONSUME** (tüketilecek slot) kuyruktaki **dolu slotların sayısını** göstermektedir. Tüketici ancak kuyrukta tüketilecek mesaj varsa tüketebilir yani bu semafor değeri sıfırdan büyükse ortak bellekten mesajı almalıdır. Başlangıçta, ilk semafor parent proses tarafından kuyruktaki toplam slot sayısına, ikinci semafor ise sıfır olarak ilklendirilmelidir.



Şekil. 2. Üretici-tüketici probleminde semaforların çapraz operasyonları.

Üretici ve tüketici prosesler paralel olarak şekil-2 deki akışa uygun olarak işleyecelerdir. Semafor üzerinde operasyon "Ele-geçir" (Acquire) : semafor değeri pozitifse azalt diğer halde bekle (blok) demektir. "Serbest-bırak" (Release) : semafor değerini atomik olarak bir arttır demektir.

Komut satırından girilen miktar adetince mesaj üretilir ve tüketilirse child prosessler sonlanır ve parent proses paylaşılmış bellek ve semaforları silerek sonlanır..

DENEYLER

1. Unix sisteminde home dizininizde lab7 dizinine appendikste(yada ekte) verilen kodları aktarınız.

   Ayrı ayrı 3 adet C kodunu çalışabilir hale getiriniz : **prod_cons_parent.c**, **producer.c** ve **consumer.c**.

2. **prod_cons_parent programını** istediğiniz mesaj sayısı parametresi ile çalıştırınız. 3 proses tarafından yazılan mesajları bir yere not ediniz. Programı kaynak kodunu inceleyip anlamaya çalışınız. Programın çalışma akışını anlamaya çalışınız ve tüm mesajların üretilip tüketildiğinden emin olunuz.

3. Hangi prosesin daha hızlı olduğunu anlamaya çalışınız (producer (üretici) veya consumer(tüketici)) ? ve sebebi anlayınız.

4. Şimdi program **consumer.c** de, MAXSLEEPTIME 'ı 10 yapınız, çalışan dosyasını üretip Adım-2 ve Adım-3 ü gerçekleştiriniz. Sonuçları analiz edip yaptığınız değişikliğin etkisini anlamaya çalışınız.

5. Adım-4 teki aynı MAXSLEEPTIME değeri ile, **producer.c ve consumer.c,** programlarında semaforların kullanıldığı kısmı siliniz (// karakteri ile yoruma dönüştürebilirsiniz), silerken semaforlarla ilgili kısmı sildiğinize dikkat ediniz. Sonra değiştirilen kodları yeniden derleyip Adım-2 ve Adım-3 ü gerçekleştiriniz. Yazılan mesajları analiz edip, sonuçları anlamaya çalışınız.

6. Kodta paylaşılmış bellek nasıl sıfırlanmıştır? Kaç semafor kullanılmıştır? Herbir semaforun amacı nedir?

APPENDIKS
(Başlangıç Kaynak Kodları)

```
/*
*****************************************************************************
* Genel başlık dosyası : common.h
*****************************************************************************
*/
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>

#define MSGNUMBER 10           /* tüketicideki farklı mesaj sayısı*/
#define SLOT_LEN  50            /* Bir mesajdaki max karakter sayısı */
#define N_SLOTS   6      /* Kuyruktaki max boş slot sayısı */
#define MAXMSGS   25       /* üretilen yada tüketilen max mesaj sayısı*/
struct MEMORY {                                 /* paylaşılmış bellek yapısı */
       char buffer [N_SLOTS][SLOT_LEN];         /* Mesajlar için array */
       int head, tail;            /* kuyruğun başı ve sonu için indeksler */
       };

struct sembuf acquire = {0, -1, SEM_UNDO}, /* Acquire operasyonu için */
              release = {0,  1, SEM_UNDO}; /* Release operasyonu için */

enum {AVAIL_SLOTS, TO_CONSUME}; /* Kümedeki elemanların Indekleri 0 ve 1 iki semafora
                                   karşılık geliyor  */


/*
*****************************************************************************
* Uretici-Tuketici problemi için Parent process
* Adapted by A.Kostin from J.S.Gray, Interprocess Communication in UNIX
*
* Process sadece bir parametreye ihtiyaş duyar : üretilen ve tüketilen mesaj sayısı
* (Not :bu parameter, common.h da tanımlı MAXMSGS den fazla olmamalı
*
*
* Dosya adı: cons_prod_parent.c
*****************************************************************************
*/

#include "common.h"

main (int argc, char *argv[])
{
static struct MEMORY sharmemory;   /* Uretici ve tuketici için ortak bellek yapisi */
static ushort init_val[2] = {N_SLOTS, 0}; /* semaforlarin baslangic degerleri */
int semid, shmid;      /* semafor ve pay. bel. icin tanımlayıcılar */
int child;
char *shmptr;                                   /* Pointer pay. bellek icin */
int prod_id, cons_id, parent_id;                /* process ID leri */
union semun arguments;         /* semaforlarin baslangic atamalari icin */

printf ("Parent process basliyor...\n");
if (argc != 2) {printf("Kullanim: %s mesaj_sayisi\n", argv[0]); exit(1); }
if (atoi(argv[1]) > MAXMSGS)   {printf("Cok fazla mesaj...\n"); exit(1);}
printf("%d mesaj uretilecek ve tuketilecek\n", atoi(argv[1]));

parent_id = getpid();                          /*Key olarak kullanılacak*/
sharmemory.head = sharmemory.tail = 0;         /* baslangic kuyruk indeksleri */

/* pay. bellek : Yarat ve bagla*/
shmid = shmget(parent_id, sizeof(sharmemory), IPC_CREAT|0666);
if (shmid < 0){perror("shmget problem"); exit(1);}
```

```
shmptr = shmat(shmid, 0, 0);
if (shmptr == (char *) -1) {perror("shmat problem"); exit(1);}

/* pay. bellek : head=tail=0 ile sıfırlama */
memcpy(shmptr, (char *)&sharmemory, sizeof(sharmemory));
printf("Paylasilmis Bellek hazir...\n");

/* semafor : yarat ve ilklendir */
semid = semget(parent_id, 2, IPC_CREAT|0666);
if (semid == -1){perror("semget problem"); exit(1);}
arguments.array = init_val;
if (semctl(semid, 0, SETALL, arguments) == -1){perror("semctl problem"); exit(1);}
printf ("Semaphorlar hazır...\n");

/* Uretici ve tuketici processleri yarat */
prod_id = fork();
if (prod_id == -1) {perror("parent:fork problem 1"); exit (1);}
if (prod_id == 0) execl("producer", "producer", argv[1], 0);

cons_id = fork();
if (cons_id == -1) {perror("parent:fork problem 2"); exit(1);}
if (cons_id == 0) execl("consumer", "consumer", argv[1], 0);

/* uretici ve tuketici processleri bekle */
printf("Parent process bekliyor...\n");
wait(0);
wait(0);

/* pay. bel. ve semaforları sil */
shmdt(shmptr);
shmctl(shmid, IPC_RMID, 0);
semctl(semid, 0, IPC_RMID, 0);
printf("\nTum processler sonlandi...\n");
exit(0);
}

/*
*****************************************************************************
* Uretici-Tuketici problemi için Uretici process
* Adapted by A.Kostin from J.S.Gray, Interprocess Communication in UNIX
*
* Process sadece bir parametreye ihtiyaç duyar : üretilecek mesaj sayısı
* (Not :bu parametre, common.h da tanımlı MAXMSGS den fazla olmamalı
*
*
* Dosya adı: producer.c
*****************************************************************************
*/

#include "common.h"

main (int argc, char *argv[])
{
static char *messages[MSGNUMBER] =
       {
       "First message", "Second message", "Third message", "Fourth message",
       "Fifth message", "Sixth message", "Seventh message", "Eighth message",
       "Ninth message", "Tenth message"
       };
static char local[SLOT_LEN];               /* bir mesaj için local array */
static struct MEMORY  *memptr;         /* yapilandirilmis pay. bel. icin pointer */
char  *shmptr;                       /* yapilandirilmamis pay. bel. icin pointer */

int myparid = getppid(); /* parent process ID, IPC icin KEY olacak*/
int semid, shmid;      /* semafor ve pay. bel. icin tanimlayicilar */
int i, j, sleeptime;
int maxmsg;                              /* Uretilen mesaj sayisi */
int curmsg;                             /* Suanki mesajın numarasi */

printf ("URETICI basladi...\n");
maxmsg = atoi(argv[1]);
```

```
srand((unsigned)getpid());   /* random sayı ureteci icin tohum */

/* pay. bel. : erisim sagla, bagla ve yapilandir */
shmid = shmget(myparid, 0, 0);
if (shmid < 0){perror("Uretici:shmget problem"); exit(1);}
shmptr = shmat(shmid, 0, 0);
if (shmptr == (char *) -1) {perror("Uretici:shmat problem"); exit(1);}
memptr = (struct MEMORY *) shmptr; /* yapilandirilmiş ortak bellege pointer */


/* semaforlar: erisim sagla */

semid = semget(myparid, 2, 0);
if (semid == -1){perror("Uretici: semget problem"); exit(1);}

/* Uretici icin ana dongu: pay. bel. ge mesajlari yerlestir */
for (curmsg = 0; curmsg < maxmsg; curmsg++)
    {
    memset(local, '\0', sizeof(local));    /* local arrayi sıfırla */
    sleeptime = rand() % 6;          /* random 5 sn. ye kadar uyu */
    sleep(sleeptime);
    i = curmsg % MSGNUMBER;          /* mesajin array indeksi */
    strcat(local, messages[i]);      /*sonraki mesaji gecici olarak local arrayinde tut */
    printf("Ureticinin hazirladigi mesaj: %s\n", local);

    /* pay. bel. bos bir slot al */
    acquire.sem_num = AVAIL_SLOTS;
    j = semop(semid, &acquire, 1);
    if (j == -1) {perror("uretici:semop problem 1"); exit(1);}

    /* local deki hazirlanmis mesajı pay. bel. de buffer a at */
    strcpy(memptr->buffer[memptr->tail], local);
    /* Move index for buffer array in shared memory */
    memptr->tail = (memptr->tail + 1) % N_SLOTS;

    /* Simdi tuketici icin enaz bir mesaj var */
    release.sem_num = TO_CONSUME;
    j = semop(semid, &release, 1);
    if (j == -1){perror("uretici:semop problem 2"); exit(1);}
    }

printf("\n Uretici sonlandı...\n");
exit(0);
}

/*
*****************************************************************************
* Uretici-Tuketici problemi için Tuketici process
* Adapted by A.Kostin from J.S.Gray, Interprocess Communication in UNIX
*
* Process sadece bir parametreye ihtiyaç duyar : üretilecek mesaj sayısı
* (Not :bu parametre, common.h da tanımlı MAXMSGS den fazla olmamalı
*
*
* Dosya adı: consumer.c
*****************************************************************************
*/

#include "common.h"
#define MAXSLEEPTIME 5
main (int argc, char *argv[])
{
static char local[SLOT_LEN];    /* bir mesaj için local array */
static struct MEMORY  *memptr; /* yapilandirilmis pay. bel. icin pointer */
char  *shmptr;       /* yapilandirilmamis pay. bel. icin pointer */

int myparid = getppid();/* parent process ID, IPC icin KEY olacak*/
int semid, shmid;  /* semafor ve pay. bel. icin tanimlayicilar */
int i, j, sleeptime;
int maxmsg;       /* Uretilen mesaj sayisi */
int curmsg;        /* Suanki mesajın numarasi */
```

```
printf ("TUKETICI basladi...\n");
maxmsg = atoi(argv[1]);
srand((unsigned)getpid());       /* random sayı ureteci icin tohum */

/* pay. bel. : erisim sagla, bagla ve yapilandir */
shmid = shmget(myparid, 0, 0);
if (shmid < 0){perror("tuketici:shmget problem"); exit(1);}
shmptr = shmat(shmid, 0, 0);
if (shmptr == (char *) -1) {perror("tuketici:shmat problem"); exit(1);}
memptr = (struct MEMORY *) shmptr; /*yapilandirilmiş ortak bellege pointer */


/* semaforlar: erisim sagla */
semid = semget(myparid, 2, 0);
if (semid == -1){perror("tuketici: semget problem"); exit(1);}

/* tuketici icin ana dongu: pay. bel. den mesajlari al */
for (curmsg = 0; curmsg < maxmsg; curmsg++)
    {
    memset(local, '\0', sizeof(local));    /* local arrayi sıfırla */
    sleeptime = rand() % MAXSLEEPTIME;    /* random MAXSLEEPTIME sn. ye kadar uyu*/
    sleep(sleeptime);

    /* Mumkunse pay. bel. sıradaki mesajı al */
    acquire.sem_num = TO_CONSUME;
    j = semop(semid, &acquire, 1);      /* pay. bel. de bir mesajı bekle */
    if (j == -1) {perror("tuketici:semop problem 1"); exit(1);}

    /* Pay. bel. ten bir mesajı local array e at */
    strcpy(local, memptr->buffer[memptr->head]);
    /* kuyruğun head indeksini arttır */
    memptr->head = (memptr->head + 1) % N_SLOTS;

    /* Uretici icin enaz bir slot bosalacak */
    release.sem_num = AVAIL_SLOTS;
    j = semop(semid, &release, 1);       /* Simdi bir slot bosaldi*/
    if (j == -1){perror("tuketici:semop problem 2"); exit(1);}
    printf(".........................Tuketici mesajı aldı: %s\n", local);
    }

printf("\n Tuketici sonlandı...\n");
exit(0);
}
```