SİSTEM PROGRAMLAMA

Deney #6- UNIX'te IPC olarak Mesaj kuyruklarını anlama

GİRİŞ

Mesaj kuyrukları IPC mekanizmalarından biridir. Bir mesaj kuyruğu bir process tarafından OS çekirdeğinde yaratırlır ve başka processlerin erişimi için tutulur. Bir mesaj kuyruğu bir anahtar (key) ile tanımlanır ve eşleştirilir. Kuyruk yoluyla haberleşecek tüm processler bu anahtarı bilmek zorundadır. Tıpkı dosyalar gibi, Mesaj kuyruğunun bir sahibi ve erişim modları vardır.

Kuyruktaki herbir mesajın bir tipi ve tiple eşleşmiş bir verisi vardır. Mesaj tipi pozitif bir tamsayıdır.

İşlev olarak mesaj kuyrukları pipe ile ortak belleğin arasındandır. Pipe'lardan daha esnek bir şekilde veriye erişim sağlar fakat bu erişim ortak bellek gibi veriye gelişigüzel erişim şeklinde olmaz. Bir mesaj kuyruğu sistemden system çağrısı **msgctl()** ile veya kabuk komutu **ipcrm** ile uzaklaştırılabilir.

Bu deneyin amacı mesaj kuyruklarını iki process arasındaki bir iletişim mekanizması olarak anlamaktır.

Verilen Kodlar da hatalar çıkarsa düzeltiniz. Geliştirdiğiniz kodları, sorulara olan cevapları ve ekran görüntülerini **bir rapor olarak hazırlayıp, teslim ediniz.**

DENEYLER

1. Takip eden C kodunu (**sndmsg.c**) derleyip birleştirin ve çalışabilir dosyasını üretin. Bu kod klavyeden <key> <tip> <text> formatında alınan bilgi doğrultusunda bir mesaj kuyruğu yaratır ve kuyruğa bir mesaj gönderir. (Derleyici struct msgbuf tipini tanıyamazsa aşağıdaki yorumu açınız.)

```
/*dosya adı : sndmsg.c*/
/* Kullanım:program_adı  anahtar tip  boşluksuz_bir_mesaj */
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

/* msgbuf veri yapısı types.h header dosyasında tanımlıdır*/
/* struct msgbuf { long mtype;
                        char mtext [100]; }; */

main (int argc, char * argv[])

{int msid, v;
 struct msgbuf mess;
 if (argc != 4) {printf ("Kullanım: <key> <tip> <text>\n"); exit (1);}
 /* mesaj kuyruğunu yaratma veya erişim sağlama */
 msid = msgget ((key_t) atoi (argv[1]), IPC_CREAT | 0666);
 if (msid == -1)     {printf("Mesaj kuyruğunu elde edemedi\n"); exit (1);}
 /* Komut satırından mesajı hazırlayalım */
 mess.mtype = atoi (argv[2]);     /*  mesaj tipi */
 strcpy (mess.mtext, argv[3]);    /*   mesajın metni          */
 /* mesajı kuyruğa gönder */
 v = msgsnd (msid,&mess,strlen(argv[3]) + 1, 0);
 if (v < 0)      printf("HATA : Mesaj kuyruğuna yazılamadı\n");
printf ("Gönderici sonlandı\n"); exit(0);
}
```

2. Takip eden C kodunu (**rcvmsg.c**) derleyip birleştirin ve çalışabilir dosyasını üretin. Bu kod basit bir şekilde klavyeden <key> <tip> formatında girilen bilgi doğrultusunda istenen mesajı kuyruktan alır ve gösterir. Son olarak mesaj kuyruğunu sistemden siler.

/*dosya adı : rcvmsg.c */

```
/* Kullanım:program_adı  anahtar tip */

/* Usage: rcvmsg   key  type */
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <errno.h>

struct msgbuf { long mtype;
            char mtext [100]; };

main (int argc, char argv[])

{int msid, v;
 struct  msgbuf mess;
 if (argc != 3) {printf ("Kullanım: <key> <tip>\n"); exit (1);}
 /* mesaj kuyruğu handle ını al */
 msid = msgget ((key_t) atoi (argv[1]), 0);
 if (msid == -1) {printf ("Bu key ile mesaj kuyruğuna erişim sağlanamadı\n"); exit (1);}
 /* kuyruktan bir mesaj al, mesaj kuyruktan silinmiş olur */
 v = msgrcv(msid, (struct msgbuf *)&mess, 100, atoi(argv[2]), IPC_NOWAIT);
 if (v < 0){if (errno == ENOMSG)
          printf("Belirtilen tipte mesaj kuyrukta yok\n");
         else     printf ("HATA:kuyruktan okunamadı\n"); }
 else printf ("[%d]  %s\n", mess.mtype, mess.mtext);
if        (msgctl(msid,IPC_RMID, 0) < 0)
          {printf(“Hata: mesaj kuyruğu sistemden silinemedi\n”); exit(1);}
else      printf(“Mesaj kuyruğu (key = %ld ) sistemden silindi\n”, atoi(argv[1]));
exit(0);
}
```

3. Gönderici processi bir çokkez (5 - 10) farklı key, mesaj ve mesaj-tipi ile çalıştırın (yarattığınız kuyrukları ve değerlerini bir kağıda not edin).

4. Alıcı processi bir çok kez çalıştırın ve sonuçları anlamaya çalışın..

5. Her iki programı (**sndmsg.c ve rcvmsg.c)** kopya ederek farklı **isimlerde (sndloop.c ve rcvloop.c)** şu şekilde değiştirin : İlk program her 5 saniyede bir mesajı mesaj-kuyruğuna atsın. İkinci program bir döngü içinde kuyruktan tüm mesajları alıp göstersin.

Not 1: Eğer tip=0 ayarlanırsa **msgrcv() sıradaki mesajı alacaktır.**

Her iki programda da processlerin ne zaman sonlanacağınıda ayarlamanız gerekir bunun için örneğin ilk program 20 mesajı kuyruğa atsın. İkinci program kuyruktaki tüm mesajları alınca **mesaj-kuyruğunu** silsin. İlk processi arkaalanda çalıştırın (bunun için % **programadı &) sonra diğerini çalıştırın parallel çalışmayı bu şekilde halletmiş olun.**

6. Her iki programda (**sndmsg.c ve rcvmsg.c)** öğrenci bilgilerini (öğrenci no, ad, şehir) tutan bir struct veri yapısı tip=1 ile temsil edilsin, metin tabanlı mesajları ise tip=2 ile temsil edilsin. Mesaj tipine göre bilgiler komut satırından alınsın (örneğin **sndmsg key tip no ad sehir)**. Basitçe öğrenci bilgilerini tutan struct veri yapısı, (char *) tipine cast edilip, kuyruk mesajı olarak düşünülsün. Bunun için her iki programda örneğin **typedef struct ogrenci { . . . } tbilgi**; gibi bir tip üzerinden çalışmalısınız.

7. Son olarak komut satırından **ipcs** ile once kuyruğun sistemde olduğuna bakın varsa **ipcrm** ile kuyruğu silin.