Sistemas Operacionais – S11

Leonardo Winter Pereira - 944424
Lucas Zimmermann Cordeiro – 944050

Comunicação Entre Processos (IPC) em UNIX

1) Programa que envie dados de um processo A para um processo B através de filas de mensagens

O Programa em questão está em anexo juntamente com este arquivo (Proj01.c). Explicações referentes ao código estão também presentes no arquivo .c

Para compilá-lo, basta executar os seguintes comandos:

```
gcc Proj01.c -o proj01 -lrt

./pro01
```

Como mencionado, este projeto enfoca um método primário para IPC, filas de mensagens.

Uma fila é uma estrutura de dados que permite atendimento FIFO (First In, First Out). Em uma fila de mensagens, a primeira mensagem que é colocada na fila é a primeira mensagem a ser lida da fila, ocorrendo um sincronismo entre origem e destino, pois as mensagens são lidas na ordem que foram enviadas. O oposto a isto é o assincronismo, onde a ordem recebida pode ser diferente da ordem enviada. Há quatro chamadas de sistema associadas com filas de mensagem:

- **msgget(...)** - é usada para criar uma fila de mensagens e/ou obter o identificador de uma fila de mensagens baseado em sua chave de sistema. A chave é um número único que identifica a fila de mensagens. Cada processo que deseja se comunicar com a fila de mensagens deve conhecer a sua chave. O identificador é um número designado pelo sistema que é obtido usando a chamada **msgget(...)**

e a chave. O identificador é um parâmetro para os outros comandos de filas de mensagens.

- **msgctl(...)** - é usado para realizar operações de controle na fila (inclusive removê-la).
- **msgsnd(...)** - é usada para colocar uma mensagem na fila.
- **msgrcv(...)** - é usada para ler uma mensagem da fila.

Filas de mensagens são relativamente simples de usar, posto que, o Sistema Operacional controla os detalhes internos de comunicação. Quando se envia uma mensagem através da fila, qualquer processo que espera por uma mensagem naquela fila é alertado. O sistema operacional verifica a integridade da fila e não permite que dois processos tenham acesso a uma fila de modo destrutivo, não sendo necessário, portanto, travar o acesso a ela. Adicionalmente, filas de mensagens constituem um excelente mecanismo para que processos troquem informações de "controle". Embora as filas de mensagens tenham estas vantagens, elas têm duas desvantagens distintas:

a. filas de mensagens são lentas em transferir grandes quantias de dados;

b. limitação no tamanho do pacote de dados que pode ser transferido; por conseguinte, filas de mensagens são melhores quando taxas lentas de transferência de dados podem são utilizadas (com "bandwidth" limitado).

2) Programa que envie dados de um processo A para um processo B através de área de Mensagem Compartilhada

O Programa em questão está em anexo juntamente com este arquivo (client.c e server.c). Explicações referentes ao código estão também presentes no arquivo .c

Para compilá-lo, basta executar os seguintes comandos:

```
gcc server.c -o server -lrt

./server

gcc client.c -o client -lrt

./server
```

Como mencionado, este projeto utiliza uma área de mensagem compartilhada para enviar mensagens entre um servidor e um cliente.

Uma mensagem compartilhada é uma memória que pode ser acessada por múltiplos programas com intenção de realizar comunicação entre eles ou para impeder cópias redundantes. Este é um meio eficiente de realizar trocas de dados entre programas.

Este método pode ser utilizado também para se comunicar entre *threads* dentro de um mesmo programa.

Os passos para conseguir criar uma área de memória compartilhada são basicamente:

1. **Criação e conexão:** Similar à outras formas de IPC, uma área de mensagem compartilhada é criada através do seguinte commando:

```
int shmget(key_t key, size_t size, int shmflg);
```

Se bem sucedido, este comando retorna um identificador para o segmento de memória. O argumento *key* deve ser criado usando *ftok()*

2. **Anexar a si mesmo no segmento:** Antes de poder utilizar a área compartilhada, é necessário se anexar utilizando o seguinte comando:

```
void *shmat(int shmid, void *shmaddr, int shmflg);
```

Desta forma, você tem um ponteiro para este segmento de memória. Note que *shmat()* retorna um ponteiro do tipo *void*, e no nosso caso, queremos tartar este ponteiro como *char.* Você pode, entretanto, tratar este ponteiro como qualquer coisa, dependendo do tipo de dado que se encontra neste segmento. Ponteiros para um *array* de estruturas aceitam esta forma de utilização.

```
data = shmat(shmid, (void *)0, 0);
```

3. **Leitura e Escrita:** Para escrever um conteúdo na memória, basta fazer:

```
printf("Enter a string: ");
gets(data);
```

E para ler:

```
printf("shared contents: %s\n", data);
```

4. **Desregistrar do segmento e deletá-lo:**

Quando você acabou de utilizar o segmento de memória, é necessário se desregistrar do mesmo e deletar este segmento, através dos seguintes comandos:

```
int shmdt(void *shmaddr);
```

shmaddr é o endereço recebido na função *shmat()*.

Para deletar o segmento, basta realizar:

```
shmctl(shmid, IPC_RMID, NULL);
```