Relatório – Comunicação Inter-processos

Marcelo Fernandes de Morais Filho
Diogo Nogueira Knop

Engenharia Eletrônica - UTFPR
03 de setembro de 2018

1. Send & Receive

Na primeira tarefa, possuímos dois arquivos com o intuito de enviar e receber mensagens via fila entre processos.

O processo mqueue-send inicialmente cria uma fila com capacidade para 10 mensagens de tamanho *int*. Esta fila possui o nome my_queue, que pode ser utilizado também por outros processos que a chamarem com o mesmo nome.

Após isso, o processo de recepção fica em loop infinito esperando mensagens nesta fila recém-criada.

Já o arquivo de envio de mensagens mqueue-send apenas abre a fila com o nome definido anteriormente (/my_queue) com permissões de escrita. Na sequência, um número aleatório entre 0 e 99 é escolhido e enviado para a fila por referência com prioridade 0. Esta mensagem é prontamente recebida no processo de leitura que aguardava recepção.

1.1 Saída:

```
Retval: 7388
Retval: 0
Received msg value 83
Sent message with value 83
Sent message with value 86
Received msg value 86
Sent message with value 77
Received msg value 77
Sent message with value 15
Received msg value 15
Sent message with value 93
Received msg value 93
Sent message with value 35
Received msg value 35
Sent message with value 86
Received msg value 86
Sent message with value 92
```

1.2 Código principal:

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

int main(int argc, char *argv[], char *envp[])
{
    int retval = fork();
    if (retval == 0)
        execve("./mqueue-send", argv, envp);
    else
        execve("./mqueue-recv", argv, envp);
}
```

Este código foi utilizado para a criação de dois processos separados, um rodando o processo de envio e o outro de recepção. Na tarefa de memória compartilhada, o mesmo procedimento foi utilizado.

2. Shared Memory

Neste exercício, foram criados dois processos para mostrar como é feito o compartilhamento de memória para troca de informações.

Primeiramente, a função shm_open() é utilizada para a abertura de uma região de memória compartilhada, com o nome *sharedmem*. O tamanho da área utilizada de memória é então ajustado e mapeado em um ponteiro que apontará para a mesma região de memória em processos que utilizarem o mesmo nome na abertura da região.

2.1 Saída

```
Wrote value 886
Read value 886
Wrote value 777
Read value 777
Wrote value 915
Read value 915
Wrote value 793
Read value 793
Wrote value 335
Read value 335
Wrote value 386
Read value 386
Wrote value 492
Read value 492
Wrote value 649
Read value 649
```

3. Conclusão

Os programas testados empregam métodos diferentes para a comunicação entre processos. Enquanto no primeiro a comunicação é feita via fila, onde podemos ter certeza que uma mensagem enviada não será perdida caso outra mensagem seja enviada antes da leitura ser feita. Isso foi verificado empiricamente ao matarmos o processo de recepção e deixarmos o de envio rodando. Após iniciarmos a recepção novamente, todas as mensagens enviadas anteriormente são imediatamente recebidas.

Já no método feito por compartilhamento de memória não possui esta restrição: ao matarmos o processo de recepção, todos os dados de envio são perdidos até a recepção ser reinicializada. Para evitarmos este problema, poderíamos fazer uma flag de reconhecimento, onde o processo de envio não escreveria novamente na região de memória até que a leitura fosse realizada.