FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Practical Assignment Nº2

Inter-Process Communication: Semaphores, Shared Memory, Message Queues; Threads;

Relatório realizador por:

David Oliveira, nº2021221929

João Cunha, nº2020252382

Pl1, Grupo 1

Unidade Curricular de Sistemas Operativos
Licenciatura em Engenharia Eletrotécnica e de Computadores – 2023/2024

Problema 1:

Neste problema é utilizado um modelo produtor-consumidor utilizando processos e filas de mensagens.

As variáveis globais incluem identificadores para as filas de mensagens e arrays para gerenciar arquivos e chaves.

A função “signal_handler” é resposável por liberar recursos quando o programa é interrompido. A função “generate_random” vai gerar números aleatórios entre 1 e 49.

A função do produtor gera números aleatórios e envia-os para a fila “msq1”, controlando o fluxo de produção através da fila “msq2”.

A função do consumidor recebe números da fila “msq1”, verifica se o número já foi gerado para a chave, se não for repetido, escreve o número no arquivo correspondente e por fim, quando uma chave está completa, o consumidor termina e sinaliza o produtor através de “msq2”.

A função main cria as filas de mensagens, define os manipuladores de sinal e cria os processos do produtor e do consumidor. O programa espera que todos os consumidores terminem e libera recursos ou seja os arquivos e as filas de mensagem.

Concluindo, esta resolução exemplifica uma aplicação de comunicação inter-processos utilizando filas de mensagens, com uma implementação robusta para a manipulação de sinais e gerenciamento de recursos.

Problema 2:

No problema 2 usamos dois programas, *client* e *server*, estes processos partilham valores *foating point* através de memória partilhada e sincronizam a leitura e escrita da memória partilhada através de dois semáforos, um para indicar a leitura e outro para indicar a escrita.

O processo *client*:

Começa por abrir o ficheiro "input.asc" e lê um valor por linha, até o fim do ficheiro, e guarda os valores por ordem num *array* tipo *double*. Ao mesmo tempo guarda numa variável quantos valores foram lidos. Após executar estas operações pode fechar o ficheiro.

Cria uma memória partilhada chamada: "my_shared_memory.bin" com permissões de leitura e escrita para todos os processos. Depois define o tamanho da memória partilhada para guardar uma variável *double*. Agora pode mapear a memória partilhada na memória virtual do processo *client.*

É necessário criar dois semáforos: "write_semaphore" e "read_semaphore", com permissões de leitura e escrita para todos os processos.

Através de um loop *for()* o processo executa, para todos os valores do *array* tipo *double*, os seguintes passos: 1.Espera pelo semáforo "write_semaphore"; 2.Escreve na memória partilhada o valor correspondente do *array* tipo *double*; 3.Envia o semáforo "read_semaphore".

Após enviar todos os valores envia o valor *DBL_MIN* para indicar que não existem mais números para enviar. Espera mais uma vez pelo semáforo "write_semaphore" para poder fechar e eliminar os dois semáforos.

Falta só desalocar o espaço de memória virtual para a memória partilhada, fechar e eliminar a memória partilhada.

No processo *server:*

Abrimos a memória partilhada: "my_shared_memory.bin ", criada pelo processo *client* para leitura e escrita e mapeamos esta em memória virtual do processo. Abrimos também os dois semáforos: "write_semaphore" e "read_semaphore", criados pelo processo *client* para leitura e escrita. Finalmente criamos um ficheiro "input.bin" em modo escrita.

Para começar a transferência de dados entre os dois processos: 1.O processo *server* envia o semáforo "write_semaphore", para o processo *client* escrever na memória partilhada; 2.Espera pelo semáforo "read_semaphore";

Entra num ciclo *while()* com a condição de ler a memória partilhada e compará-la com o valor *DBL_MIN.* Se forem iguais o ciclo *while()* termina e é enviado uma última vez o semáforo "read_semaphore", terminar a comunicação. São fechados os dois semáforos.

Dentro do ciclo *while()* o programa escreve o valor da memória lida no ficheiro "input.bin" e volta a executar as operações 1. e 2. descritas a cima.

Agora o processo pode desalocar o espaço de memória virtual para a memória partilhada, e fechar a mesma. Fechar também o ficheiro “input.bin”.

Depois volta a abrir o ficheiro “input.bin”, desta vez, em modo leitura. Lê todos os números escritos anteriormente, em ordem, para um *array* tipo *double.*

Cria um ficheiro “output.asc” em modo escrita e através de um loop *for()* multiplica cada valor do *array* por “4.0” e escreve no ficheiro “output.asc” todos as variáveis do *array*. Para terminar fecha o ficheiro “output.asc”.

Ambos os processos, *client* e *server*, possuem *signal handlers* para os sinais: *SIGHUP; SIGTERM; SIGINT; SIGQUIT* e *SIGKILL*. São redirecionados para a função “cat_SIGNAL”, onde são fechados, desalocados e eliminados os ficheiros, semáforos e memória partilhada necessários.

Problema 3:

Neste programa o objetivo é receber dois inteiros introduzidos pelo utilizador no terminal, criar um *thread* enviar os inteiros. O *thread* criado deve somar e multiplicar os inteiros recebidos e enviar o resultado das duas operações e terminar. O "main()" deve esperar que o *thread* termine, deverá receber os valores enviados pelo *thread*, escrever o resultado no terminal e terminar o processo.

No código são incluídas as librarias "<stdio.h>", "<stdlib.h>", "<pthread.h>"e "<signal.h>". É inicializado uma variável global ponteiro para inteiro *result*, para ser usado pelo "main()" e pelas funções de S*ignal Handler*.

O "main()" começa por declarar uma variável tipo "pthread_t" para guardar o id do *thread* a ser criado. Declara também uma variável ponteiro *void* para guardar o resultado enviado pelo *thread*.

Para guardar os dois inteiros são criados duas variáveis *int*, um ponteiro para indicar para cada uma das variáveis *int* e uma tabela com dois ponteiros, um para cada ponteiro anterior. É escrito no terminal "Choose two integers:" através de "printf()", usamos "scanf()" duas vezes e guardamos os dois inteiros.

É criado um *thread* através de "pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start)(void *), void *arg)", se for devolvido um inteiro diferente de 0 é ativado o "perror()" e o programa termina. É passado para o novo *thread* o ponteiro para a tabela de ponteiros criada anteriormente com *type cast* para *void*.

Dentro do *thread* é criada uma variável ponteiro para a tabela de ponteiros, para guardar o argumento recebido do "main()" com *type cast* para *int*(inteiro). É alocada memória através de "malloc()" para guardar os dois inteiros a ser retornados ao "main()", sem desaparecer quando o *thread* terminar.

São realizadas a operações matemáticas de somar e multiplicar entre os inteiros recebidos. E através de "pthread_exit()" é enviado o ponteiro para a memória alocada para o "main()" e o *thread* termina.

Após criado o *thread* são chamados *signal handlers* através de "signal" para "SIGHUP", "SIGTERM", "SIGINT", "SIGQUIT" e "SIGKILL". Assim, se o processo receber um destes sinais irá desalocar a memória criada pelo *thread* e terminar sem deixar memória alocada que não pode ser mais usada.

A seguir espera pelo termino do *thread* e o ponteiro para um *array* com os resultados das operações. Escreve o resultado no terminal através de "printf()". Desaloca a memória criada pelo *thread* e termina.