Universidade Federal De Uberlândia
Faculdade de Computação
Sistemas Operacionais
GSI018

EULLER HENRIQUE BANDEIRA OLIVEIRA
11821BSI210

Relatório :
Atividade Prática de Fixação
(Unidade 3)

Uberlândia
2021

Computador utilizado:

- Notebook: Lenovo Ideapad S145
- Processador: i7-8565U
- Cpus: 8
- Ram: 20gb
- Ssd: 256gb
- Hd: 1tb
- Placa de vídeo 1: Intel UHD Graphics 620 (Whiskey Lake)
- Placa de vídeo 2: NVidia GM108M (GeForce MX110)
- So1: Windows 10 Home
- So2: Deepin 20.2
- So utilizado: So2

PARTE 1 (P1, P2, P3 E P4):
A variável sem_t do tipo mutex é declarada.
A variável global go do tipo int é declarada. Tal variável recebe o valor 0.

A struct shared_area é declarada. A struct possui as seguintes variáveis: sem_t mutex, pid_t pidd[4], int queue[10], int end, int size e int stop.

O ponteiro shared_area_ptr do tipo shared_area é declarado.

PRODUTOR:

Uma memória compartilhada é criada, isso faz com que os valores da struct shared_area sejam compartilhados entre processos.

Um semáforo é criado:

```
sem_init (& sem , pshared , valor );
sem_init((sem_t *)&shared_area_ptr->mutex,1,1)
```

Escopo Interprocess -> Quando pshared é diferente de zero, o semáforo pode ser compartilhado por outros processos.

CREATE_P1_P4:

Os processos 1 ,2, 3 e 4 são criados. O id de cada processo é armazenado no vetor compartilhado pidd.

EXEC_P1_P3:

Se o processo atual for o processo 1, 2 ou 3, a função queue_push é chamada.

queue_push:

Um while infinito se inicia:

Se a variável compartilhada stop for igual à 0:

A função sem_wait é chamada:

```
sem_wait ((sem_t*)&shared_area_ptr->mutex)
```

Down: (SEMÁFARO VERDE) -> (SEMÁFARO VERMELHO)

Se a região crítica já estiver sendo utilizada por um processo, o acesso à região crítica é bloqueado para os próximos processos e esses processos são colocados em uma fila de espera.

A função srand é chamada, tal função cria uma semente para a geração de números randômicos. Nesse caso, a semente é o processo atual * o tempo atual * rand().

Se a fila estiver vazia (se o tamanho da fila for igual à 0):

Um número randômico entre 1 e 1000 é inserido na posição 0 da variável compartilhada queue.
A variável compartilhada end recebe o valor 0.
A variável compartilhada size é incrementada.

Se a fila não estiver vazia (se o tamanho da fila for menor que 10):

Um número randômico entre 1 e 1000 é inserido na posição presente na variável compartilhada end + 1.
A variável compartilhada size é incrementada.

Se a fila não estiver vazia (se o tamanho da fila for menor que 10):

Um printf irá exibir o id do processo atual, o tamanho da fila e o número inserido.

Se a fila estiver cheia (se o tamanho da fila for igual a 10) e o processo atual for o processo 1:

Um printf irá exibir o id do processo atual, o tamanho da fila e o número inserido.

Um printf irá informar que a fila está cheia.

Um printf irá informar que o processo 1 irá enviar um sinal para o processo 4.

A variável compartilhada stop receberá o valor 1, com isso, a execução do processo 1,2 3 será interrompida.

Um sinal (SIGUSR1) será enviado para o processo p4 por meio da função kill.

A função sem_post é chamada:

sem_post((sem_t*)&shared_area_ptr->mutex)

Up: (SEMÁFARO VERMELHO) -> (SEMÁFARO VERDE)
O processo que estiver no começo da fila de espera poderá acessar a região crítica

EXEC_P4:

Se o processo atual for o processo 4:

Um semáforo é criado:

sem_init (& sem , pshared , valor);

sem_init(&mutex, 0,1);

Escopo Intraprocesso -> Quando pshared é igual a zero, o semáforo pode ser compartilhado pelas threads do processo atual

A variável thread_id do tipo pthread é criada.

A thread_1 é criada por meio da função pthread_create.

Um while infinito se inicia:

A função signal chama a função call_thread_2.
A função pause faz com que a execução do processo seja pausada até um sinal ser recebido.

thread_1:

O pipe_01 é criado por meio da função mkfifo.

Um while infinito se inicia:

Se go for igual a 1:

A função queue_pop é chamada e o nome do pipe criado é enviado para essa função.

thread_2:

O pipe_02 é criado por meio da função mkfifo.

Um while infinito se inicia:

Se go for igual a 1:

A função queue_pop é chamada e o nome do pipe criado é enviado para essa função.

Se go for igual a 2:

O while se encerra, ou seja, com isso, a função é finalizada e o processo 4 volta a esperar um sinal do processo 1

call_thread_2:

Se o sinal for igual a SIGUSR1:

Um printf exibe informa que o processo 4 recebeu o sinal do processo 1
A variável global g1 recebe o valor 1. Com isso, a função queue_pop é chamada na thread 1 e a função queue_pop será chamada após a thread_02 ser chamada.
A função thread 2 é chamada.

queue_pop:

A função sem_wait é chamada:

sem_wait (&mutex)

Down: (SEMÁFARO VERMELHO) -> (SEMÁFARO VERDE)

O processo que estiver no começo da fila de espera poderá acessar a região crítica

Se o tamanho da fila for maior que 0:

A variável num é declarada. Tal variável recebe o valor presente no início da fila, ou seja, o valor que será removido.

O tamanho da fila é decrementado.

Um laço for vai de 0 ao tamanho da fila:

O elemento presente no vetor compartilhado queue com o índice i é substituído pelo elemento presente no vetor compartilhado com o índice i+1, ou seja, o elemento posterior ao elemento inicial se torna o elemento inicial.

Lógica do for:

queue[0] = queue[1]
queue[1] = queue[2]
queue[2] = queue[3]
queue[3] = queue[4]
queue[4] = queue[5]
queue[5] = queue[6]
queue[6] = queue[7]
queue[7] = queue[8]
queue[8] = queue[9]
queue[9] = queue[10]

Um printf informa o id da thread atual, o tamanho da fila e o número retirado.

A variável fd do tipo int é declarada. Tal variável recebe o retorno da função open. A função open abre no modo escrita o pipe recebido pela função queue_pop.

A função write é chamada, tal função possui como argumento a variável fd, o endereço da variável num e o tamanho do tipo int.

A função close é chamada. Tal função fecha o pipe que foi aberto.

Se a fila estiver vazia (se o tamanho da fila for igual a 0):

A variável compartilhada stop recebe o valor 0. Com isso, p1, p2 e p3 volta a inserir valores na fila.

A variável global go recebe o valor 2. Com isso, a thread_1 é finalizada, p4 volta a esperar um sinal e a thread_2 para de remover valores da fila.

A função sem_post é chamada:

sem_post(&mutex)

Up: (SEMÁFARO VERDE) -> (SEMÁFARO VERMELHO)

Se a região crítica já estiver sendo utilizada por um processo, o acesso à região crítica é bloqueado para os próximos processos e esses processos são colocados em uma fila de espera.

MAIN:

A variável global shared_area_ptr recebe o retorno da função produtor.

O vetor pidd pertencente à struct compartilhada shared_area é zerado.

A variável size pertencente à struct compartilhada shared_area é zerado.

A variável stop pertencente à struct compartilhada shared_area é zerado.

A variável global go é zerada.

Os processos 1, 2, 3 e 4 são criados por meio da função CREATE_P1_P4

Um while interrompe o fluxo do código até que os processos 1,2 e 3 sejam criados.

Os processos 1,2,3 são executados

Um while interrompe o fluxo do código até que o processo 4 seja criado.

O processo 4 é executado

PARTE 2 (P5,P6,P7):

O código trava após os 10 valores serem inseridos na fila, isso ocorre pois a função queue_pop não é chamada.