Programação Sistema com Semáforos

Sincronização entre Processos

- Cada processo criado no sistema é independente, e desde que se encontre no estado Preparado pode ser selecionado para correr no processador.
- No entanto, se uma aplicação for composta por vários processos estes podem ter de interagir uns com os outros de forma a trabalharem de forma sincronizada.
- Iremos ver a seguir algumas destas situações.

Exemplo: Dois Processos Partilham o Ecrã

- Neste programa são criados dois processos filho. Cada um escreve no ecrã várias linhas de texto. Deve notar-se que de cada vez corre um dos processos filho.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>

int main(){
  int pid, childPid;

  pid=fork();
  if( pid == 0 ){

    for ( int i = 0; i < 1000; i++ )
                printf( "Filho 11111111111111111111\n" );

  }
  else{

    pid=fork();
    if( pid == 0 ){

      for ( int i = 0; i < 1000; i++ )
                printf( "Filho 22222222222222222222\n" );

    }
    else{

      wait(&childPid);
      wait(&childPid);
    }

  }

  return 0;
}
```

Sincronização de Processos com Semáforos

- O funcionamento dos processos pode ser sincronizado através de semáforos. Pode-se por exemplo utilizar um semáforo binário para deixar apenas um processo avançar para a escrita no ecrã, e só depois deste terminar a escrita, deixar avançar o outro (exclusão mútua). Deve ser utilizada a opção -pthread na compilação do programa: gcc -o prog prog.c -pthread.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <sys/mman.h>

int main(){
  int pid, childPid;

  //A variável do semáforo tem de ser única, e partilhada pelos filhos
  //Para isso tem de ser criada uma variável partilhada através da função mmap
  //Não pode ser uma cópia para cada um dos filhos... é o que acontece com as variáveis normais...
  sem_t *sem = mmap(NULL, sizeof (sem_t*), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
```

```
  //O semáforo é inicializado
  sem_init(sem, 1, 1); //O terceiro parâmetro é 1, significa que deixa passar 1 vez e fica a 0

  pid=fork();
  if( pid == 0 ){

    //Wait sobre o semáforo
    sem_wait( sem );

    for ( int i = 0; i < 1000; i++ )
                printf( "Filho 11111111111111111111\n" );

    //Garante-se que tudo o que foi escrito é actualizado no ecrã
    fflush(stdout);

    //Post sobre o semáforo
    sem_post( sem );

  }
  else{

    pid=fork();
    if( pid == 0 ){

      //Wait sobre o semáforo
      sem_wait( sem );

      for ( int i = 0; i < 1000; i++ )
                printf( "Filho 22222222222222222222\n" );

      //Garante-se que tudo o que foi escrito é actualizado no ecrã
      fflush(stdout);

      //Post sobre o semáforo
      sem_post( sem );

    }
    else{

      wait(&childPid);
      wait(&childPid);

      //O semáforo criado na memória partilhada é destruido
      sem_destroy(sem);

      //A zona de memória partilhada é destruída
      munmap(sem, sizeof(sem_t*));

    }

  }

  return 0;
}
```

- Igual ao anterior, acrescentando-se apenas verificações sobre as operações realizadas com o semáforo. No caso de erro é fornecida uma mensagem, e a seguir terminado o programa através da função exit.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <sys/mman.h>
#include <stdlib.h>

int main(){
  int pid, childPid;

  //A variável do semáforo tem de ser única, e partilhada pelos filhos
  //Para isso tem de ser criada uma variável partilhada através da função mmap
  //Não pode ser uma cópia para cada um dos filhos... é o que acontece com as variáveis normais...
  sem_t *sem = mmap(NULL, sizeof (sem_t), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);

  //O semáforo é inicializado
  if( sem_init(sem, 1, 1) == -1){
    printf("Erro na criação do semáforo\n");
```

```
    exit(1);
  }

  pid=fork();
  if( pid == 0 ){

    //Wait sobre o semáforo
    if( sem_wait( sem ) == -1 ){
      printf("Erro no wait do semáforo\n");
      exit(1);
    }

    for ( int i = 0; i < 1000; i++ )
                printf( "Filho 111111111111111111111111\n" );

    //Garante-se que tudo o que foi escrito é actualizado no ecrã
    fflush(stdout);

    //Post sobre o semáforo
    if( sem_post( sem ) == -1 ){
      printf("Erro no post do semáforo\n");
      exit(1);
    }


  }
  else{

    pid=fork();
    if( pid == 0 ){

      //Wait sobre o semáforo
       if( sem_wait( sem ) == -1 ){
         printf("Erro no wait do semáforo\n");
         exit(1);
      }

      for ( int i = 0; i < 1000; i++ )
                printf( "Filho 222222222222222222222222\n" );

      //Garante-se que tudo o que foi escrito é actualizado no ecrã
      fflush(stdout);

      //Post sobre o semáforo
      if( sem_post( sem ) == -1 ){
        printf("Erro no post do semáforo\n");
        exit(1);
      }

    }
    else{

      wait(&childPid);
      wait(&childPid);

      //O semáforo criado na memória partilhada é destruido
      if ( sem_destroy(sem) == -1 ){
        printf( "Erro ao destruir o semáforo\n");
        exit( 1 );
      }

      //A zona de memória partilhada é destruída
      if (munmap(sem, sizeof(sem_t)) < 0) {
        perror("Erro na destruição do semáforo");
        exit(1);
      }
    }

  }

  return 0;
}
```

Outro Exemplo: Dois Processos Partilham uma Variável

- Apresenta-se a seguir um outro exemplo de partilha de recursos entre processos, neste caso uma variável. Cada um dos processos vai incrementar uma variável partilhada 100.000 vezes. Era de esperar que no final

a variável contivesse o valor 200.000 mas tal não acontece. Protegendo a zona do programa onde é incrementada a variável com recurso a um semáforo já é obtido o resultado esperado... Porquê?

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <sys/mman.h>

int main(){
  int pid, childPid;

  //A variável do semáforo tem de ser única, e partilhada pelos filhos
  //Para isso tem de ser criada uma variável partilhada através da função mmap
  //Não pode ser uma cópia para cada um dos filhos... é o que acontece com as variáveis normais...
  sem_t *sem = mmap(NULL, sizeof (sem_t), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);

  //É também criada uma variável inteira partilhada
  int *n = mmap(NULL, sizeof (int), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);

  //O semáforo é inicializado
  sem_init(sem, 1, 1);

  *n = 0;

  pid=fork();
  if( pid == 0 ){

    for ( int i = 0; i < 100000; i++ ){
      //Wait sobre o semáforo
      //sem_wait( sem );

      (*n)++;
      printf( "Filho 11111111111111111111: %d\n", *n );
      fflush(stdout);

      //Post sobre o semáforo
      //sem_post( sem );
    }

  }
  else{

    pid=fork();
    if( pid == 0 ){

      for ( int i = 0; i < 100000; i++ ){

        //Wait sobre o semáforo
        //sem_wait( sem );

        (*n)++;
        printf( "Filho 22222222222222222222: %d\n", *n );
        fflush(stdout);

        //Post sobre o semáforo
        //sem_post( sem );
      }

    }
    else{

      wait(&childPid);
      wait(&childPid);

      printf("%d\n", *n);
      fflush(stdout);

      //O semáforo é destruido
      sem_destroy(sem);

      //As zonas de memória partilhadas são destruídas
      munmap(sem, sizeof(sem_t));
      munmap(n, sizeof(int));
    }
```

```
  }

  return 0;
}
```

Ainda Outro Exemplo: Dois Processos Comunicam Através de uma Variável

- No exemplo seguinte dois processos filhos trocam informação através de uma variável. Um dos processos coloca sucessivamente valores nessa variável, e o outro processo retira os valores dessa variável. Neste caso os processos também precisam de estar sincronizados pois caso contrário o segundo processo não conseguirá obter todos os valores. Veja o que acontece sem sincronização entre os processos, e depois retire os comentários das linhas do programa que sincronizam os processos.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <sys/mman.h>

int main(){
  int pid, childPid;

  //semáforo que indica se já pode ser colocado um novo número na variável partilhada
  sem_t *sem_put = mmap(NULL, sizeof (sem_t), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);

  //semáforo que indica se já pode ser retirado um novo número da variável partilhada
  sem_t *sem_get = mmap(NULL, sizeof (sem_t), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);

  //variável partilhada
  int *n = mmap(NULL, sizeof (int), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);

  //Os semáforos são inicializado
  sem_init(sem_put, 1, 1);//A variável está livre, pode ser colocado um número
  sem_init(sem_get, 1, 0);//A variável ainda não contém um número para ser retirado

  *n = 0;

  pid=fork();
  if( pid == 0 ){

    for ( int i = 0; i < 10; i++ ){

      //Wait sobre o semáforo que controla a inserção de novos números
      //sem_wait( sem_put );

      *n = i;

      printf( "Filho 1111111111111111111111: %d\n", *n );
      fflush(stdout);

      //Post sobre o semáforo que controla a obtenção de novos números
      //sem_post( sem_get );
    }

  }
  else{

    pid=fork();
    if( pid == 0 ){

      for ( int i = 0; i < 10; i++ ){

        //Wait sobre o semáforo que controla a obtenção de novos números
        //sem_wait( sem_get );

        printf( "Filho 2222222222222222222222: %d\n", *n );
        fflush(stdout);

        //Post sobre o semáforo que controla a inserção de novos números
        //sem_post( sem_put );
      }

    }
    else{
```

```
      wait(&childPid);
      wait(&childPid);

      //Os semáforos são destruidos
      sem_destroy(sem_put);
      sem_destroy(sem_get);

      //As zonas de memória partilhadas são destruídas
      munmap(sem_put, sizeof(sem_t));
      munmap(sem_get, sizeof(sem_t));
      munmap(n, sizeof(int));
    }

  }

  return 0;
}
```

Último Exemplo: Produtor-Consumidor com Dois Processos

- Um dos processos (produtor) coloca números num buffer (vector) que tem uma quantidade limitada de posições. Quando enche o buffer tem de esperar que o outro processo (consumidor) retire números do buffer e liberte posições que possam ser preenchidas com novos números. Podem ser adicionados novos processos produtores e consumidores. Este é um exemplo clássico de sincronização entre processos.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <sys/mman.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>

#include
#include
#include

#define MAX_NUM 10
#define TOTAL_NUM 20
#define MAX_TIME_PROD 2
#define MAX_TIME_CONS 4

struct SharedStruct{
  int v[MAX_NUM];
  int in;
  int out;
};

int main(){
  int pid, childPid;

  //semáforo que indica se já pode ser colocado um novo número na variável partilhada
  sem_t *sem_put = mmap(NULL, sizeof (sem_t), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);

  //semáforo que indica se já pode ser retirado um novo número da variável partilhada
  sem_t *sem_get = mmap(NULL, sizeof (sem_t), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);

  //semáforo que apenas permite que um processo utilize a estrutura partilhada de cada vez
  sem_t *sem_mutex = mmap(NULL, sizeof (sem_t), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);

  //variável partilhada
  struct SharedStruct *pSharedStruct = mmap(NULL, sizeof (struct SharedStruct), PROT_READ | PROT_WRITE, MAP_SHARED |

  //Os semáforos são inicializado
  sem_init(sem_put, 1, MAX_NUM);//No início podem ser colocados MAX_NUM números
  sem_init(sem_get, 1, 0);//No início não pode ser retirado nenhum número
  sem_init(sem_mutex, 1, 1);//Apenas um processo pode utilizar a estrutura de dados de cada vez

  //Inicializa a estrutura partilhada
  pSharedStruct->in=0;
  pSharedStruct->out=0;
  for( int i = 0; i < MAX_NUM; i++ )
    pSharedStruct->v[i] = 0;
```

```
pid=fork();
if( pid == 0 ){

  srand(time(NULL));

  for ( int i = 0; i < TOTAL_NUM; i++ ){

    //Simulamos que o produtor demora algum tempo a gerar o número...
    int t = rand() % MAX_TIME_PROD;
    sleep( t );

    //Wait sobre o semáforo que controla a inserção de novos números
    sem_wait( sem_put );

    pSharedStruct->v[pSharedStruct->in] = i;
    printf( "Produtor colocou: %d\n", pSharedStruct->v[pSharedStruct->in] );
    //printf( "Produtor demorou: %d\n", t );
    fflush(stdout);
    pSharedStruct->in++;
    if(pSharedStruct->in == MAX_NUM )
      pSharedStruct->in = 0;

    //Post sobre o semáforo que controla a obtenção de novos números
    sem_post( sem_get );
  }

}
else{

  pid=fork();
  if( pid == 0 ){

    srand(time(NULL) + 100);

    for ( int i = 0; i < TOTAL_NUM; i++ ){

      //Simulamos que o consumidor demora algum tempo a ir retirar o número...
      int t = rand() % MAX_TIME_CONS;
      sleep( t );

      //Wait sobre o semáforo
      sem_wait( sem_get );

      printf( "Consumidor retirou: %d\n", pSharedStruct->v[pSharedStruct->out] );
      //printf( "Consumidor demorou: %d\n", t );
      fflush(stdout);

      pSharedStruct->out++;
      if(pSharedStruct->out == MAX_NUM )
      pSharedStruct->out = 0;

      //Post sobre o semáforo
      sem_post( sem_put );
    }

  }
  else{

    wait(&childPid);
    wait(&childPid);

    //Os semáforos são destruidos
    sem_destroy(sem_put);
    sem_destroy(sem_get);
    sem_destroy(sem_mutex);

    //As zonas de memória partilhadas são destruídas
    munmap(sem_put, sizeof(sem_t));
    munmap(sem_get, sizeof(sem_t));
    munmap(sem_mutex, sizeof(sem_t));

    munmap(pSharedStruct, sizeof(struct SharedStruct));
  }

}
```

```
    return 0;
}
```

Última revisão efectuada em 15/01/18