Sistemas Operativos 2019/20

Segundo Trabalho Prático

Fumadores

Realizado por: 93015-Daniel Gomes e 93236-André Morais

Introdução

Introdução

Este trabalho prático tem em vista a compreensão dos mecanismos associados à execução e sincronização de processos e threads.

O problema associado é como fazer com que as entidades que necessitam dos recursos os usem nas alturas certas sem que a entidade geradora de recursos faça a notificação direta às entidades gastadoras quando os pacotes de que estes necessitam estiverem completos e sem que as entidades gastadores tenham de estar a fazer verificações desnecessárias para comprovar se já existem os recursos necessários.

O programa em questão tem em vista a gestão de memória partilhada por três intervenientes: ***Agent***, ***Watcher e Smoker.*** Usando 3 tipos de produtos (papel, tabaco e fósforos), existem 3 fumadores (3 ***smokers***) mas cada um deles tem uma fonte inesgotável de um destes recursos, necessitando apenas dos outros 2. Um agente (***agent***) produz recursos em pacotes de 2 recursos distintos, ou seja, sempre que o agente produz um pacote há um (e só um) fumador que pode fumar.A entidade Agent apenas pode iniciar uma nova produção de outros 2 recursos quando o fumador correto já tiver recolhido os ingredientes do pacote anterior.Cada uma das entidades possui alguns estados consoante a ação que estão a realizar.

Implementação do Problema

Para a implementação deste problema temos como base a utilização de semáforos e memória partilhada. No total, são utilizados 7 semáforos,sendo que um deles funciona em exclusão mútua, ***mutex****.* A razão da utilização destes é bastante simples se tivermos em conta que, como foi referido anteriormente, possuímos 3 entidades para a qual são partilhadas entre estas algumas informações, pelo que precisamos de garantir que não haverá conflitos quer na mudança de estados quer na distribuição de ingredientes. Assim, com a presença de semáforos garantimos que apenas uma entidade de cada vez, entra nas zonas de memória partilhada.

Inicialmente, foi fornecido pelo Professor algum código base para ser completado:

- ***semSharedMemAgent.c***
- ***semSharedMemWatcher.c***
- ***semSharedMemSmokers.c***

Dentro destes 3 ficheiros já se encontravam definidas as zonas onde seria necessário introduzir código, além de que as regiões críticas, onde atua o mutex, também já estavam implementadas. Caso ocorra um erro na utilização do semáforo de exclusão mútua,ou seja no acesso às zonas de memória partilhada, será impressa no terminal uma mensagem de erro a indicar isso mesmo.As estruturas associadas à criação dos semáforos, a definição das constantes a ser utilizadas já nos eram fornecidas também.

Na tabela a seguir representada encontra-se todas as atualizações das atividades a serem realizadas pelas entidades além de onde são utilizados os semáforos necessários para garantir o funcionamento correto.

Semáforo	Entidade		Ação		Função	
	Up()	Down()	Up()	Down()	Up()	Down()
mutex	Todas	Todas	Entrar na zona crítica	Sair da zona crítica	Todas	Todas
waitCigarette	Smoker	Agent	Indica que o smoker x acabou de enrolar	Indica que o agent está à espera que o smoker acabe de enrolar	rollingCigarette()	waitForCigarette()
ingredient	Agent	Watcher	Indicar que o ingredient[x] foi produzido	Indica que o ingredient[x] foi usado	prepareIngredient() closeFactory()	waitForIngredient()
wait2Ings	Watcher	Smoker	Indica que o Smoker está à espera do ingrediente x	Indica que o ingrediente já não é procurado pelo smoker.	waitForIngredient() informSmoker()	waitForIngredient()

Nesta tabela, o x corresponde a um elemento do vetor ingredient[], wait2Ings[] ou para identificar um respetivo smoker.

Por fim, apresenta-se agora todos os estados,já definidos pelo Professor, das entidades em causa, estados estes que se encontram no ficheiro ***probConst.h.*** A apresentação destes estados são fundamentais para a compreensão do código implementado.

```
/* Agent state constants */

/** \brief agent initial state, preparing pack of 2 ingredients */
#define  PREPARING         1
/** \brief agent waiting for smoker to finish rolling cigarette */
#define  WAITING_CIG       2
/** \brief agent is closing factory */
#define  CLOSING_A         3
```

Estados do Agente

```
/* Wachers state constants */

/** \brief watcher waiting for ingredient (each watcher is responsible for a
different ingredient) */
#define  WAITING_ING       0
```

```
/** \brief watcher updating reservations */
#define  UPDATING         1
/** \brief watcher informing smoker that he can start rolling */
#define  INFORMING        2
/** \brief watcher is closing */
#define  CLOSING_W        3
```

Estados dos Watchers

```
/* Smokers state constants */

/** \brief smoker is waiting for the 2 missing ingredients  */
#define  WAITING_2ING     0
/** \brief smoker is rolling cigarette  */
#define  ROLLING          1
/** \brief smoker is smoking  */
#define  SMOKING          2
/** \brief smoker is closing  */
#define  CLOSING_S        3
```

Estados dos Smokers

Agent

prepareIngredients()

```
static void prepareIngredients ()
{
   if (semDown (semgid, sh->mutex) == -1) {
/*enter critical region */
       perror ("error on the up operation for semaphore access (AG)");
       exit (EXIT_FAILURE);
   }
     sh->fSt.st.agentStat=PREPARING;
     saveState(nFic,&sh->fSt);

     int firstIng=0;
     int secondIng=0;

     firstIng=rand()%3;
     bool first=true;
     while(first || firstIng==secondIng){
         secondIng=rand()%3;
         first=false;
     }
     sh->fSt.ingredients[firstIng]++;
     sh->fSt.ingredients[secondIng]++;

   if (semUp (semgid, sh->mutex) == -1) {
/* leave critical region */
       perror ("error on the up operation for semaphore access (AG)");
       exit (EXIT_FAILURE);
   }
     if(semUp(semgid,sh->ingredient[firstIng])==-1){
       perror ("error on the up operation for semaphore access (AG)");
       exit (EXIT_FAILURE);
     }
     if(semUp(semgid,sh->ingredient[secondIng])==-1){
       perror ("error on the up operation for semaphore access (AG)");
       exit (EXIT_FAILURE);
     }
}
```

O ***Agent*** tem a possibilidade de preparar os 3 ingredientes, mas apenas 2 de cada vez. Esta função começa por atualizar o estado do ***Agent*** para “***Preparing***”,dentro da

zona de exclusão mútua, o que indica que 2 produtos estão a ser produzidos; Gera o primeiro ingrediente e, enquanto o segundo ingrediente não tiver sido produzido ou enquanto os dois ingredientes forem iguais, então será gerado um novo segundo ingrediente. Com os dois ingredientes produzidos, são incrementados uma unidade aos índices correspondentes a cada um dos ingredientes.

Por fim, ao sair da zona de exclusão mútua, os semáforos de cada um dos ingredientes (***ingredient[firstIng]*** e ***ingredient[secondIng]***) são atualizados através de ***SemUp()*** para que possam ser verificados pelos ***Watchers***.

waitForCigarette()

```
static void waitForCigarette ()
{
   if (semDown (semgid, sh->mutex) == -1) {
/* enter critical region */
       perror ("error on the up operation for semaphore access (AG)");
       exit (EXIT_FAILURE);
   }
   sh->fSt.st.agentStat=WAITING_CIG;
   saveState(nFic,&sh->fSt);
   if (semUp (semgid, sh->mutex) == -1) {
/* leave critical region */
       perror ("error on the up operation for semaphore access (AG)");
       exit (EXIT_FAILURE);
   }
    if (semDown(semgid, sh->waitCigarette)==-1){
       perror ("error on the up operation for semaphore access (AG)");
       exit (EXIT_FAILURE);
    }
}
```

A função ***waitForCigarette()*** apenas anuncia que o ***Agent*** está à espera que o Smoker acabe de enrolar o cigarro, mudando o estado do ***Agent*** para “***Waiting_Cig”***. Como se pretende que o Agent espere, efetua-se também um ***semDown()*** no semáforo ***waitCigarrete***.

closeFactory()

```
static void closeFactory ()
{
   if (semDown (semgid, sh->mutex) == -1) {
/* enter critical region */
       perror ("error on the up operation for semaphore access (AG)");
       exit (EXIT_FAILURE);
   }

   sh->fSt.st.agentStat=CLOSING_A;
    saveState(nFic, &sh->fSt);
   sh->fSt.closing=1;

   if (semUp (semgid, sh->mutex) == -1) {
/* leave critical region */
       perror ("error on the up operation for semaphore access (AG)");
       exit (EXIT_FAILURE);
   }

   //alterar o semaforo dos watchers
   if(semUp(semgid, sh->ingredient[TOBACCO])==-1){
       perror ("error on the up operation for semaphore access (AG)");
       exit (EXIT_FAILURE);
   }
   if(semUp(semgid, sh->ingredient[MATCHES])==-1){
       perror ("error on the up operation for semaphore access (AG)");
       exit (EXIT_FAILURE);
   }
   if(semUp(semgid, sh->ingredient[PAPER])==-1){
       perror ("error on the up operation for semaphore access (AG)");
       exit (EXIT_FAILURE);
   }
   /* TODO: insert your code here */
}
```

Esta função tem como objetivo alertar os ***Watchers*** de que o ***Agent*** deixou de produzir. Assim, o estado do ***Agent*** é alterado para "***Closing_A***" e a ***flag*** associada ao encerramento do agente, ***closing***, passa a ter o valor de 1. De seguida, alerta-se então todos os ***Watchers***, efetuando ***semUp()*** a cada um dos semáforos dentro do *array* ***ingredient***. As constantes ***TOBACCO,MATCHES,PAPER*** contêm o indice correpondente a cada um dos ingredientes.

Watcher

waitForIngredient()

```
static bool waitForIngredient(int id)
{
   bool ret=true;

   if (semDown (semgid, sh->mutex) == -1)  {
/* enter critical region */
       perror ("error on the up operation for semaphore access (WT)");
       exit (EXIT_FAILURE);
   }

   sh->fSt.st.watcherStat[id]=WAITING_ING;
   saveState(nFic,&sh->fSt);

   /* TODO: insert your code here */
   if (semUp (semgid, sh->mutex) == -1) {
/* exit critical region */
       perror ("error on the down operation for semaphore access (WT)");
       exit (EXIT_FAILURE);
   }
   if  (semDown(semgid,sh->ingredient[id])==-1){
       perror("error on the down operation for semaphore (WT)");
       exit (EXIT_FAILURE);
   }

   /* TODO: insert your code here */
   if (semDown (semgid, sh->mutex) == -1 )  {
/* enter critical region */
       perror ("error on the up operation for semaphore access (WT)");
       exit (EXIT_FAILURE);
   }
   if(sh->fSt.closing){
       sh->fSt.st.watcherStat[id]=CLOSING_W;
       saveState(nFic,&sh->fSt);
       ret=false;
   }
   /* TODO: insert your code here */
   if (semUp (semgid, sh->mutex) == -1) {
/* exit critical region */
       perror ("error on the down operation for semaphore access (WT)");
       exit (EXIT_FAILURE);
   }
   if(sh->fSt.closing){
        if  (semUp(semgid,sh->wait2Ings[id])==-1){
```

```
            perror("error on the up operation for semaphore access (WT)");
        exit(EXIT_FAILURE);
     }
    }
    return ret;
}
```

Com este método pretende-se esperar com que o ***Watcher*** que o ***Agent*** produza os ingredientes,sendo passado como argumento o id do Watcher em causa. Assim, começa-se por atualizar o estado do ***Watcher*** em causa (tendo em conta o ***id***) para "***WAITING_ING***". Saindo da primeira região crítica efetua-se a atualização do semáforo respectivo ao ***Watcher***, através de ***SemDown()***, com vista a esperar pelo ***Agent***. Entrando na segunda região crítica, caso a flag closing contenha o valor de 1, então atualizamos o estado do ***Watcher*** para "***CLOSING_W***" e variável booleana ***ret***, retornada por esta função, passa a conter o valor de falsidade. Por fim, ao sair da segunda região crítica, caso a ***flag closing*** esteja com valor 1, efetua-se a atualização do semáforo ***wait2Ings*** com ***SemUp()***, de notar que esta última condição poderia estar dentro da última região crítica.

updateReservations()

```
static int updateReservations (int id)
{
   int ret = -1;

   if (semDown (semgid, sh->mutex) == -1)  {
/* enter critical region */
       perror ("error on the up operation for semaphore access (WT)");
       exit (EXIT_FAILURE);
   }
   sh->fSt.st.watcherStat [id]=UPDATING;
   saveState(nFic,&sh->fSt);
   //reserve the quantity of the agent ingredient in cause
   sh->fSt.reserved[id]=sh->fSt.ingredients[id];

   //Returns the id
   if (sh->fSt.reserved[HAVETOBACCO]>0 && sh->fSt.reserved[HAVEPAPER]>0){
       sh->fSt.reserved[HAVETOBACCO]--;
       sh->fSt.reserved[HAVEPAPER]--;
       ret=MATCHES;
   }
   if(sh->fSt.reserved[HAVETOBACCO]>0 && sh->fSt.reserved[HAVEMATCHES]>0) {
       sh->fSt.reserved[HAVETOBACCO]--;
```

```
        sh->fSt.reserved[HAVEMATCHES]--;
        ret=PAPER;
    }
    if(sh->fSt.reserved[HAVEMATCHES]>0 && sh->fSt.reserved[HAVEPAPER]>0){
        sh->fSt.reserved[HAVEMATCHES]--;
        sh->fSt.reserved[HAVEPAPER]--;
        ret =TOBACCO;
    }
    /* TODO: insert your code here */

    if (semUp (semgid, sh->mutex) == -1) {
/* exit critical region */
        perror ("error on the down operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }

    return ret;
}
```

Esta função tem o intuito de, tendo em conta a quantidade de recursos disponíveis nas reservas, retorna o índice do ***Smoker*** que pode começar a enrolar o cigarro para depois o fumar. Assim, como ponto de partida, o estado do ***Watcher*** em causa (tendo em conta o **id**) passa a ser "**UPDATING**", e reservamos a quantidade de ingredientes que o Agente produziu através da linha **`sh->fSt.reserved[id]=sh->fSt.ingredients[id].`**

Após as reservas terem sido efetuadas, efetuamos as condições necessárias para garantir que é retornado o índice do ingrediente correto. Por exemplo, caso estejam reservadas quantidades de tabaco e papel superiores a zero então retornamos o índice associado ao ***Smoker*** que possui apenas fósforos, diminuindo, como consequência, as quantidades dos produtos usados das reservas. As constantes ***HAVETOBACCO,HAVEMATCHES e HAVEPAPER*** correspondem ao **id** do fumador que contêm sempre tabaco,fósforos ou papel,respetivamente.Assim reserved, é um array em que cada índice corresponde a um ingrediente reservado pelo Watcher em causa.

informSmoker()

```
static void informSmoker (int id, int smokerReady)
{
   if (semDown (semgid, sh->mutex) == -1)  {
/* enter critical region */
       perror ("error on the up operation for semaphore access (WT)");
       exit (EXIT_FAILURE);
   }
   sh->fSt.st.watcherStat [id]=INFORMING;
   saveState(nFic,&sh->fSt);
   /* TODO: insert your code here */

   if (semUp (semgid, sh->mutex) == -1) {
/* exit critical region */
       perror ("error on the down operation for semaphore access (WT)");
       exit (EXIT_FAILURE);
   }
   if(semUp(semgid, sh->wait2Ings[smokerReady]) == -1){
       perror ("error on the up operation for semaphore access (WT)");
       exit (EXIT_FAILURE);
   }
   /* TODO: insert your code here */
}
```

Como última função da entidade ***Watchers***, a função ***informSmoker()*** tem como único objetivo informar o ***Smoker*** correto que pode começar a enrolar. Assim, atualiza-se, mais uma vez, o estado do Watcher para "***INFORMING***" e por fim, efetua-se ***semUp()*** ao semáforo respetivo ao fumador em causa, ***wait2Ings[smokerReady].***

Smoker

waitForIngredients()

```
static bool waitForIngredients (int id)
{
   bool ret = true;
   if (semDown (semgid, sh->mutex) == -1)  {
/* enter critical region */
       perror ("error on the up operation for semaphore access (SM)");
       exit (EXIT_FAILURE);
   }
   sh->fSt.st.smokerStat[id]=WAITING_2ING;// WAITING_2ING;
   saveState(nFic,&sh->fSt);
   /* TODO: insert your code here */
   if (semUp (semgid, sh->mutex) == -1) {
/* exit critical region */
       perror ("error on the down operation for semaphore access (SM)");
       exit (EXIT_FAILURE);
   }
   if(semDown(semgid,sh->wait2Ings[id])==-1){
        perror("error on the down operation for semaphore access (SM)");
       exit(EXIT_FAILURE);
    }
   /* TODO: insert your code here */
   if (semDown (semgid, sh->mutex) == -1)  {
/* enter critical region */
       perror ("error on the up operation for semaphore access (SM)");
       exit (EXIT_FAILURE);
   }
   //fechar
   if(sh->fSt.closing==1){
       sh->fSt.st.smokerStat[id]=CLOSING_S;
       saveState(nFic,&sh->fSt);
       ret=false;
    }
   /* TODO: insert your code here */
   if (semUp (semgid, sh->mutex) == -1) {
/* exit critical region */
       perror ("error on the down operation for semaphore access (SM)");
       exit (EXIT_FAILURE);
   }
```

```
    return ret;
}
```

Nesta função são esperados os dois produtos em falta. A variável ***ret*** identifica se os ingredientes estão disponíveis (true) ou se o smoker está a acabar a sua atividade. O estado do smoker começa por ser atualizado a "***WAITING_2ING***", o que indica que está recetivo aos ingredientes em falta. Para este mesmo efeito, o valor do semáforo ***wait2Ings[id]*** é modificado através de ***semDown()***, sendo ***id*** um índice passado como argumento desta função. Por fim, assim que os ingredientes estiverem produzidos e a fábrica fechada, então o estado do smoker passa a ser também "***CLOSING_S***" e à variável ***ret*** é atribuído o valor "***false***"***.*** Além disso, é importante referir que, apesar de ter sido referido nos comentários do professor que esta função deveria fazer a atualização ao inventário de ingredientes, colocamos essa funcionalidade, por opção pessoal, na função que segue esta.

rollingCigarette()

```
static void rollingCigarette (int id)
{
   double rollingTime = 100.0 + normalRand(30.0);
   if (semDown (semgid, sh->mutex) == -1)  {
/* enter critical region */
       perror ("error on the up operation for semaphore access (SM)");
       exit (EXIT_FAILURE);
   }
   sh->fSt.st.smokerStat[id]=ROLLING;
   saveState(nFic,&sh->fSt);
   //recolher os pacotes..
     if (id==HAVETOBACCO){
       //fumador 1
       sh->fSt.ingredients[HAVEMATCHES]-=1;
       sh->fSt.ingredients[HAVEPAPER]-=1;
   } else if(id == HAVEMATCHES){
       //fumador 2
       sh->fSt.ingredients[HAVETOBACCO]-=1;
       sh->fSt.ingredients[HAVEPAPER]-=1;
   } else {
       //fumador 3
       sh->fSt.ingredients[HAVETOBACCO]-=1;
       sh->fSt.ingredients[HAVEMATCHES]-=1;
   }
   if(rollingTime>0)
       usleep(rollingTime);
   /* TODO: insert your code here */
```

```
   if (semUp (semgid, sh->mutex) == -1) {
/* exit critical region */
       perror ("error on the down operation for semaphore access (SM)");
       exit (EXIT_FAILURE);
   }
   //alertar o agente que ja acabou de enrolar
   if( semUp(semgid,sh->waitCigarette) ==-1){
       perror ("error on the down operation for semaphore access (SM)");
       exit (EXIT_FAILURE);
   }
   /* TODO: insert your code here */
}
```

Esta é a função responsável por fazer os Smokers enrolar os cigarros assim que tiverem os 2 ingredientes em falta. No ínicio é gerado um tempo aleatório para que sirva de tempo usado a enrolar o cigarro. Logo de seguida é atualizado o estado do Smoker para "***ROLLING***" e, como em todas as outras atualizações de estado, é feito o ***saveState()*** para que o estado seja, efetivamente guardado. Ainda dentro da zona crítica, para cada um dos fumadores - identificados pela possessão do ingrediente que lhe é infinitamente presente - os dois ingredientes que lhe são dados são decrementados por 1, sinónimo de que foram utilizados. Seguidamente, no caso de o ***rollingTime*** ser maior do que 0, então é suspensa a execução do programa por ***rollingTime*** milissegundos. Agora fora da zona crítica, é atualizado o semáforo relativo ao cigarro enrolado (***waitCigarette***) de modo a que o agente seja alertado que o smoker acabou de enrolar.

smoke()

```
static void smoke(int id)
{
   double smokingTime;
   if (semDown (semgid, sh->mutex) == -1)  {
/* enter critical region */
       perror ("error on the up operation for semaphore access (SM)");
       exit (EXIT_FAILURE);
   }
   sh->fSt.st.smokerStat[id]=SMOKING;
   saveState(nFic,&sh->fSt);
   smokingTime=100+normalRand(30.0);
   sh->fSt.nCigarettes[id]++;
   if (semUp (semgid, sh->mutex) == -1) {
/* exit critical region */
       perror ("error on the down operation for semaphore access (SM)");
       exit (EXIT_FAILURE);
   }
```

```
    if (smokingTime>0)
        usleep(smokingTime);
    /* TODO: insert your code here */
}
```

Função responsável por atualizar o número de cigarros fumados pelo fumador correto e o estado do fumador para "***SMOKING***" além de suspender a atividade do programa durante um número aleatório de milissegundos regido por ***smokingTime*** - indicativo de que o fumador está a fumar o cigarro.

Conclusão

Execução do Programa

O programa é executado compilando primeiro todos os ficheiros na pasta ***src*** através do comando "**make all_bin**" e correndo, de seguida, o executável ***./probSemSharedMemSmokers*** da pasta **run**.

Testes

Os nossos testes consistiram em correr o ficheiro ***probSemSharedMemSmokers*** - sem ocorrência de *deadlocks* - e em correr 1 vez cada combinação de make, cmparando cada uma com o resultado do **make all_bin**.Através do executável run.sh também foi possivel correr 1000 vezes o ficheiro referido anteriormente, na qual conseguimos verificar que os resultados obtidos foram os espectáveis. Como ferramentas de debugging, foi utilizado o comando ***gdb smoker core***, que nos permite aceder ao mapa de memória do computador no preciso momento que foi executado o programa, ferramenta esta que nos foi recomendada pelo Professor.

Exemplo, assumindo que nos encontramos da pasta ***src*:**

$ make all_bin ; cd ../run

$ ./run.sh 1 > teste.txt

Para "limpar" semáforos existentes no programa pode-se executar o ***script*** na pasta ***run*** :

$ .clean.sh

Os resultados estão no ficheiros de texto do *zip* que foi enviado, juntamente com o relatório e as pastas ***src***, ***doc*** e ***run***.