universidade de aveiro

Sistemas Operativos
Prof. Nuno Lau

FUMADORES

Trabalho 2

Rui Fernandes Nmec: 92952
Alexandre Rodrigues Nmec: 92951

A participação é repartida de igual forma.

2019/2020

Index

Introdução

O presente trabalho prático, realizado no âmbito da disciplina “Sistemas Operativos”, tem como objetivo uma melhor compreensão dos mecanismos associados à execução e sincronização de processos e threads. O seu tema é a gestão de recursos que envolve vários fumadores com necessidade distinta de fumar. Foi-nos proposto completar o código fonte.

Existem três tipos de recursos: **tabaco**, **papel** e **fósforos**.
E três tipos de entidades:

Agente - 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.

Watcher - responsável por verificar se após a emissão de um novo ingrediente (um dos elementos do pacote produzido pelo agente) há algum fumador que possa fumar. Existe um watcher por cada tipo de ingrediente, ou seja, existem 3 watchers cada um especializado no seu ingrediente.

Fumador – consome os recursos.

Abordagem usada para a execução do trabalho

Inicialmente, para uma melhor compreensão do código disponibilizado, procedemos à análise de todas as funções para compreender o que cada uma realizava.
Decidimos então criar uma tabela como suporte para verificar o comportamento de cada entidade, isto é, onde e quando seria necessário ocorrer **ups** e **downs** de cada semáforo presente no ficheiro ***sharedDataSync.h***

Semáforo	***Entidade***		***Ação***		**Função**	
	Up()	***Down()***	***Up()***	***Down()***	***Up()***	***Down()***
Mutex	Todas	Todas	Sair da região critica	Entrar na região critica	Quase Todas	Quase Todas
WAITCIGARETTE	Smoker	Agent	Informa o agent que o smoker acabou de enrolar o cigarro.	O agent espera que o smoker acabe de enrolar o cigarro.	rollingCigarette()	waitForCigarette()
INGREDIENTS[ID]	Agent	Watcher	Após serem gerados 2 ingredientes notifica os watchers respetivos, ou quando a fábrica fecha, notifica todos os watchers	O watcher espera que o agent notifique a disponibilidade do ingrediente ou o fecho da fábrica.	prepareIngredients() closeFactory()	waitForIngredient()
WAIT2INGS[ID]	Watcher	Smoker	Informa o smoker que pode enrolar, ou que a fábrica está a fechar.	O smoker espera que o watcher informe a possibilidade de enrolar, ou o fecho da fábrica.	waitForIngredient() informSmoker()	waitForIngredient()

Tab.1 Tabela resumida do comportamento dos semáforos.

Comportamento de cada entidade

Como cada entidade tinha funções a serem completadas, todas as etapas e alterações realizados estão documentadas.

Agent

Função prepareIngredients

Função responsável por preparar os ingredientes necessários para o problema.

O estado do agente é atualizado para **PREPARING** e após serem gerados 2 ingredientes aleatoriamente é atualizado o stock dos mesmos.

Seguidamente o agent informa os watchers associados aos ingredientes gerados que podem proceder com o seu funcionamento.

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

    int ing1 = (rand() % NUMINGREDIENTS);
    int ing2;
    while( (ing2 = ( rand() % NUMINGREDIENTS) ) == ing1){};

    sh->fSt.ingredients[ing1]++;
    sh->fSt.ingredients[ing2]++;

    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the up operation for semaphore access (AG)");
        exit (EXIT_FAILURE);
    }
    /* TODO: insert your code here */
    semUp(semgid, sh->ingredient[ing1]);
    semUp(semgid, sh->ingredient[ing2]);
}
```

Fig.1 Função prepareIngredients

Função waitForCigarette

O estado do agent é atualizado para **WAITING_CIG** e de seguida é efetuado um down() no semáforo waitCigarette para que o agent interrompa o seu funcionamento e espere pela notificação do smoker quando este enrolar um cigarro.

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

    /* TODO: insert your code here */
    sh->fSt.st.agentStat = WAITING_CIG;
    saveState(nFic, &sh->fSt);

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

    /* TODO: insert your code here */
    semDown(semgid, sh->waitCigarette);
}
```

Fig.2 Função waitForCigarette

Função closeFactory

Faz com que o agente atualize o seu estado para **CLOSING_A** e atualiza o estado da fábrica para fechado, através do *sh->fSt.closing*.

De seguida informa todos os watchers á cerca do fecho da fábrica, através dos semáforos ingredient[id].

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

    /* TODO: insert your code here */
    sh->fSt.st.agentStat = CLOSING_A;
    sh->fSt.closing = true;
    saveState(nFic, &sh->fSt);

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

    /* TODO: insert your code here */
    semUp(semgid, sh->ingredient[0]);
    semUp(semgid, sh->ingredient[1]);
    semUp(semgid, sh->ingredient[2]);
}
```

Fig.3 Função closeFactory

Watcher

Função waitForIngredient

Primeiramente o estado do watcher é atualizado para **WAITING_ING** e é efetuado um down() no semáforo ingredient[id] para que o watcher aguarde notificação do agent.

Após ser notificado, verifica se a fábrica está a fechar, caso esteja muda o seu estado para **CLOSING_W,** avisa o smoker a si associado e muda o retorno da função para falso, interrompendo o funcionamento do watcher.

Caso contrário o valor de retorno continua verdadeiro e o watcher prossegue com o seu funcionamento.

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

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

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

    return ret;
}
```

Fig.4 Função waitForIngredients

Função updateReservations

Esta funçao tem como objetivo atualizar o array *reserved* e verificar se algum smoker pode começar a enrolar.

Inicialmente é atualizado o estado do watcher para **UPDATING,** e é reservado o ingrediente correspondente ao mesmo, através do array *reserved*.

De seguida verifica-se quantos ingredientes já se encontram reservados, caso estejam 2, então o valor de retorno da função (ret) é alterado para o id do smoker que consegue fumar utilizando os 2 ingredientes (o array *reserved* é também reposto a zeros).

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

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

    /* TODO: insert your code here */
    sh->fSt.st.watcherStat[id] = UPDATING;
    saveState(nFic, &sh->fSt);

    sh->fSt.reserved[id] = 1;

    int reserved_count = 0;
    for(int i = 0; i<NUMINGREDIENTS; i++){
        if(sh->fSt.reserved[i] == 1)
            reserved_count++;
    }

    if (reserved_count==2){
        for(int j = 0; j<NUMINGREDIENTS; j++){
            if (sh->fSt.reserved[j] == 0)
                ret = j;
            sh->fSt.reserved[j] = 0;
        }
    }

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

    return ret;
}
```

Fig.5 Função updateReservations

Função informSmoker

A função começa por atualizar o estado do watcher para **INFORMING**.

Posteriormente o watcher notifica o smoker que pode fumar (identificado pela variavél *smokerReady*) através de um up() no semáforo *wait2Ings*.

```
static void informSmoker (int id, int smokerReady)
{
    if (semDown (semgid, sh->mutex) == -1)  {
        perror ("error on the up operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.watcherStat[id] = INFORMING;
    saveState(nFic, &sh->fSt);


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

    /* TODO: insert your code here */
    semUp(semgid, sh->wait2Ings[smokerReady]);
}
```

Fig.6 Função informSmoker

Smoker

Função waitForIngredients

Esta função tem como objetivo fazer com que o smoker espere pelos 2 ingredientes que precisa. Assim o seu estado é atualizado para **WAITING_2ING** e posteriormente é feito um down() no semáforo **WAITING_2ING** para que o smoker aguarde a notificação do watcher para enrolar o cigarro.

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

    if (semDown (semgid, sh->mutex) == -1)  {
        perror ("error on the up operation for semaphore access (SM)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.smokerStat[id] = WAITING_2ING;
    saveState(nFic, &sh->fSt);

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


    /* TODO: insert your code here */
    semDown(semgid, sh->wait2Ings[id]);

    if (semDown (semgid, sh->mutex) == -1)  {
        perror ("error on the up operation for semaphore access (SM)");
        exit (EXIT_FAILURE);
    }
```

Após o semaforo ser desbloqueado, o smoker vai verificar a natureza da notificação do watcher, ou seja, avaliar se a fabrica está a fechar ou se pode enrolar um cigaro.

Caso a fábrica esteja a fechar, o estado do smoker é atualizado para **CLOSING_S** e o valor retornado pela função é falso terminando o funcionamento do smoker.

Caso contrário, o retorno continua verdadeiro, os ingredientes disponibilizados são decrementados e o smoker prossegue para a função rollingCigarette.

```
/* TODO: insert your code here */
if(sh->fSt.closing==true){
    ret = false;
    sh->fSt.st.smokerStat[id] = CLOSING_S;
    saveState(nFic, &sh->fSt);
} else{
    for(int i = 0; i<=2; i++){
        if(i!=id){
            sh->fSt.ingredients[i]--;
        }
    }
}

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

return ret;
```

Fig.7 Função waitForIngredients

Função rollingCigarette

Esta função é responsável por simular a preparação do cigarro.

É gerado um rollingTime semi-aleatório, o estado do smoker é atualizado para **ROLLING**, e então espera durante o rollingTime gerado (através do **usleep**).

Seguidamente o smoker notifica o agent que já acabou de enrolar o cigarro, fazendo up() no semáforo waitCigarette, e segue para a função smoke.

```
static void rollingCigarette (int id)
{
    double rollingTime;
    while( (rollingTime = 100.0 + normalRand(30.0)) < 0){};

    if (semDown (semgid, sh->mutex) == -1)  {
        perror ("error on the up operation for semaphore access (SM)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.smokerStat[id] = ROLLING;
    saveState(nFic, &sh->fSt);

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

    /* TODO: insert your code here */
    usleep(rollingTime);
    semUp(semgid, sh->waitCigarette);
}
```

Fig.8 Função rollingCigarette

Função smoke

Função responsável por simular o ato de fumar o cigarro, primeiramente é gerado um smokingTime semi-aleatório que vai ser usado de forma similar ao rollingTime da função anterior.

O estado do smoker é atualizado para **SMOKING** e o número de cigarros fumados por ele é incrementado.

De seguida o smoker espera através do **usleep** e volta à função waitForIngredients.

```
static void smoke(int id)
{
    double smokingTime;
    while( (smokingTime = 100.0 + normalRand(30.0)) < 0){};

    if (semDown (semgid, sh->mutex) == -1)  {
        perror ("error on the up operation for semaphore access (SM)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.smokerStat[id] = SMOKING;
    sh->fSt.nCigarettes[id]++;
    saveState(nFic, &sh->fSt);

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

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

Fig.9 Função smoke

Testes realizados para validar a solução

Primeiramente testámos a existencia de *deadlocks* no nosso programa, para tal, com auxílio do script **run.sh** , corremos o programa 1 milhão de vezes no total, com sequências de 100 mil vezes seguidas. Visto que nunca ocorreu um *deadlock* neste teste, estamos bastante seguros da consistência do nosso programa.

```
Run n.º 100000
                    Smokers - Description of the internal state

 AG  W00 W01 W02  S00 S01 S02  I00 I01 I02  C00 C01 C02
  1    0   0   0    0   0   0    0   0   0    0   0   0
  1    0   0   0    0   0   0    0   0   0    0   0   0
  2    0   0   0    0   0   0    1   1   0    0   0   0
  2    1   0   0    0   0   0    1   1   0    0   0   0
  2    1   0   0    0   0   0    1   1   0    0   0   0
  2    1   1   0    0   0   0    1   1   0    0   0   0
  2    1   1   0    0   0   0    1   1   0    0   0   0
  2    0   2   0    0   0   0    1   1   0    0   0   0
  2    0   0   0    0   0   0    1   1   0    0   0   0
  2    0   0   0    0   0   1    0   0   0    0   0   0
  2    0   0   0    0   0   2    0   0   0    0   0   1
  1    0   0   0    0   0   2    0   0   0    0   0   1
  2    0   0   0    0   0   2    0   1   1    0   0   1
  2    0   0   1    0   0   2    0   1   1    0   0   1
  2    0   1   1    0   0   2    0   1   1    0   0   1
  2    0   2   0    0   0   2    0   1   1    0   0   1
  2    0   0   0    1   0   2    0   0   0    0   0   1
  2    0   0   0    1   0   0    0   0   0    0   0   1
  2    0   0   0    2   0   0    0   0   0    1   0   1
  1    0   0   0    2   0   0    0   0   0    1   0   1
  2    0   0   0    2   0   0    1   1   0    1   0   1
  2    0   1   0    2   0   0    1   1   0    1   0   1
  2    1   1   0    2   0   0    1   1   0    1   0   1
  2    2   0   0    2   0   0    1   1   0    1   0   1
  2    0   0   0    2   0   1    0   0   0    1   0   1
  2    0   0   0    0   0   1    0   0   0    1   0   1
  2    0   0   0    0   0   2    0   0   0    1   0   2
  1    0   0   0    0   0   2    0   0   0    1   0   2
  2    0   0   0    0   0   2    1   0   1    1   0   2
  2    1   0   0    0   0   2    1   0   1    1   0   2
  2    1   0   1    0   0   2    1   0   1    1   0   2
  2    0   0   2    0   0   2    1   0   1    1   0   2
  2    0   0   0    0   1   2    0   0   0    1   0   2
  2    0   0   0    0   1   0    0   0   0    1   0   2
  2    0   0   0    0   2   0    0   0   0    1   1   2
  1    0   0   0    0   2   0    0   0   0    1   1   2
  2    0   0   0    0   2   0    1   1   0    1   1   2
  2    0   1   0    0   2   0    1   1   0    1   1   2
  2    1   1   0    0   2   0    1   1   0    1   1   2
  2    2   0   0    0   2   0    1   1   0    1   1   2
  2    0   0   0    0   2   1    0   0   0    1   1   2
  2    0   0   0    0   0   1    0   0   0    1   1   2
  2    0   0   0    0   0   2    0   0   0    1   1   3
  3    0   0   0    0   0   2    0   0   0    1   1   3
  3    3   0   0    0   0   2    0   0   0    1   1   3
  3    3   3   0    0   0   2    0   0   0    1   1   3
  3    3   3   3    0   0   2    0   0   0    1   1   3
  3    3   3   3    3   0   2    0   0   0    1   1   3
  3    3   3   3    3   3   2    0   0   0    1   1   3
  3    3   3   3    3   3   0    0   0   0    1   1   3
  3    3   3   3    3   3   3    0   0   0    1   1   3
rui@rui-VivoBook-S15-X510UF:~/Desktop/SO/Trabalho 2/semaphore_smokers/run$
```

Fig.10 Teste de deadlocks

Seguidamente, verificamos se os resultados obtidos tinham lógica face ao objetivo do projeto, se de facto as entidades funcionam como é pretendido. Este teste foi realizado manualmente, verificando linha a linha se as atualizações e mudanças de estado fazem sentido, isto foi feito para vários casos individuais, sendo que aqui demonstramos apenas um.

```
rui@rui-VivoBook-S15-X510UF:~/Desktop/SO/Trabalho 2/semaphore_smokers/run$ ./probSemSharedMemSmokers
                    Smokers - Description of the internal state

 AG  W00 W01 W02  S00 S01 S02  I00 I01 I02  C00 C01 C02
  1    0   0   0    0   0   0    0   0   0    0   0   0
  1    0   0   0    0   0   0    0   0   0    0   0   0
  2    0   0   0    0   0   0    0   1   1    0   0   0
  2    0   0   0    0   0   0    0   1   1    0   0   0
  2    0   0   0    0   0   0    0   1   1    0   0   0
  2    0   0   0    0   0   0    0   1   1    0   0   0
  2    0   0   1    0   0   0    0   1   1    0   0   0
  2    0   1   1    0   0   0    0   1   1    0   0   0
  2    0   2   0    0   0   0    0   1   1    0   0   0
  2    0   0   0    1   0   0    0   0   0    0   0   0
  2    0   0   0    2   0   0    0   0   0    1   0   0
  1    0   0   0    2   0   0    0   0   0    1   0   0
  2    0   0   0    2   0   0    0   1   1    1   0   0
  2    0   1   0    2   0   0    0   1   1    1   0   0
  2    0   1   1    2   0   0    0   1   1    1   0   0
  2    0   1   1    0   0   0    0   1   1    1   0   0
  2    0   0   2    0   0   0    0   1   1    1   0   0
  2    0   0   0    1   0   0    0   0   0    1   0   0
  2    0   0   0    2   0   0    0   0   0    2   0   0
  1    0   0   0    2   0   0    0   0   0    2   0   0
  2    0   0   0    2   0   0    1   1   0    2   0   0
  2    0   1   0    2   0   0    1   1   0    2   0   0
  2    1   1   0    2   0   0    1   1   0    2   0   0
  2    1   0   0    0   0   0    1   1   0    2   0   0
  2    2   0   0    0   0   0    1   1   0    2   0   0
  2    0   0   0    0   0   1    0   0   0    2   0   0
  2    0   0   0    0   0   2    0   0   0    2   0   1
  1    0   0   0    0   0   2    0   0   0    2   0   1
  2    0   0   0    0   0   2    1   1   0    2   0   1
  2    1   0   0    0   0   2    1   1   0    2   0   1
  2    1   1   0    0   0   2    1   1   0    2   0   1
  2    0   2   0    0   0   2    1   1   0    2   0   1
  2    0   0   0    0   0   0    1   1   0    2   0   1
  2    0   0   0    0   0   1    0   0   0    2   0   1
  2    0   0   0    0   0   2    0   0   0    2   0   2
  1    0   0   0    0   0   2    0   0   0    2   0   2
  2    0   0   0    0   0   2    0   1   1    2   0   2
  2    0   1   0    0   0   2    0   1   1    2   0   2
  2    0   0   1    0   0   2    0   1   1    2   0   2
  2    0   0   2    0   0   2    0   1   1    2   0   2
  2    0   0   0    1   0   2    0   0   0    2   0   2
  2    0   0   0    1   0   0    0   0   0    2   0   2
  2    0   0   0    2   0   0    0   0   0    3   0   2
  3    0   0   0    2   0   0    0   0   0    3   0   2
  3    3   0   0    2   0   0    0   0   0    3   0   2
  3    3   0   3    2   0   0    0   0   0    3   0   2
  3    3   3   3    2   0   0    0   0   0    3   0   2
  3    3   3   3    2   0   3    0   0   0    3   0   2
  3    3   3   3    2   3   3    0   0   0    3   0   2
  3    3   3   3    0   3   3    0   0   0    3   0   2
  3    3   3   3    3   3   3    0   0   0    3   0   2
rui@rui-VivoBook-S15-X510UF:~/Desktop/SO/Trabalho 2/semaphore_smokers/run$
```

Fig.11 Teste de funcionamento

Conclusão

A realiação deste projeto permitiu-nos aprofundar os nossos conhecimentos sobre os mecanismos associados à execução e sincronização de processos e threads.

Este trabalho foi feito com base na análise do código fornecido, sendo que grande parte do conhecimento do mesmo foi adquirido nas aulas teóricas e práticas previamente.

De modo geral, o maior desafio foi perceber todas as funções necessárias para a implementação do projeto, visto que todo o código teria que ter uma estrutura e um funcionamento correto.

Finalmente, todos os testes realizados foram bem sucedidos.