FUMADORES

Sistemas Operativos

Pedro Bastos - 93150
Universidade de Aveiro

Índice

Introdução

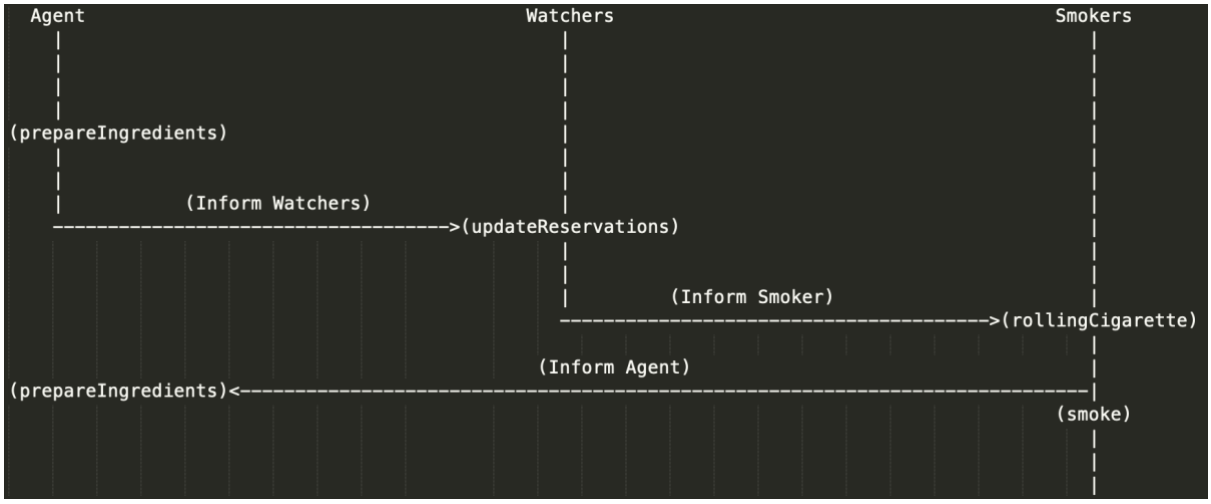
Foi pedido aos alunos que desenvolvessem uma aplicação em C que fizesse a gestão de recursos de vários fumadores com necessidades distintas de fumar. Este problema tem como principal objetivo a melhor compreensão dos mecanismos associados à execução e sincronização de processos e threads.

O docente forneceu um código base onde apenas teria de alterar as funções de 3 entidades: Agent, Watcher e Smoker. Basicamente, o agente irá produzir 2 ingredientes dos 3 possíveis (tabaco, papel e fósforos). Cada um dos 3 smokers tem um ingrediente inesgotável. Assim, o smoker que fuma com os ingredientes produzidos pelo agente será o que possui o ingrediente que falta. Os watchers (3) funcionam como intermédio entre o agente e os smokers, havendo um watcher por ingrediente.

Todas estas 3 entidades são processos independentes, pelo que devem comunicar através de semáforos e memória partilhada.

Análise inicial

Inicialmente é necessário perceber os ciclos de vida de cada uma das entidades e o ciclo conjunto. Assim, procurei entender o que cada entidade fazia e como é que interagiam entre si, que está brevemente ilustrado no seguinte diagrama:



Temos 3 estruturas base: STAT (estados de cada entidade), FULL_STAT (contém a STAT e variáveis gerais) e SHARED_DATA (contém FULL_STAT e o semáforos). Para mudar um estado de uma entidade, usa-se 'sh->fSt.st.<entidade>Stat = <estado>'. Para alterar variáveis gerais, usa-se 'sh->fSt.<variável>'. A função 'saveState(nFic,&sh->fSt)' é usada sempre que se altera um estado. Downs no mutex entra na zona crítica e ups sai da mesma.

Análise dos semáforos

Para um mais fácil entendimento do problema, realizei este trabalho por partes. Graças ao código fornecido, é possível testar apenas uma das entidades usando '$make <entidade>'. Assim, subdivide-se o problema em vários mais pequenos, o que ajuda bastante na ultrapassagem dos deadlocks. Para entender quais semáforos usar e onde usá-los, realizei uma tabela sugerida pelo professor da aula prática que me ajudou bastante a solucionar o problema:

	UP		**Down**	
	Entidade	Função	Entidade	Função
mutex	All	All	All	All
ingredient[id]	Agent	prepareIngredients() closeFactory()	Watcher	waitForIngredient()
waitCigarrete	Smoker	rollingCigarette()	Agent	waitForCigarette()
wait2Ings[id]	Watcher	waitForIngredient() informSmoker()	Smoker	waitForIngredients()

Entidades

Agent

O agente irá produzir os ingredientes necessários, 2 de cada vez. Enquanto produz, encontra-se no estado de PREPARING. Seguidamente fica no estado WAITING_CIG até que o smoker acabe de enrolar o cigarro.

- prepareIngredients()

Aqui o agente irá produzir 2 ingredientes escolhidos de forma aleatória. Dentro da zona crítica, muda o estado para PREPARING. Depois, gera 2 números aleatórios entre 0,1,2 para serem usados como os ingredientes a produzir. Faz ainda update à variável global ingredientes para atualizar o stock. Finalmente, fora da zona crítica, faz semUp no semáforo ingrediente[id] para notificar os watchers correspondentes.

```
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; //changing state
    saveState(nFic, &sh->fSt); //saving state

    int ing1 = 0; //first ingredient
    int ing2 = 0; //second ingredient

    //generating random numbers
    int lower = 0; //lower bound
    int upper = 2; //upper bound
    ing1 = (rand() % (upper - lower + 1)) + lower;
    do{

        ing2 = (rand() % (upper - lower + 1)) + lower;

    }while(ing1 == ing2); //cannot be equal numbers

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


    /* TODO: insert your code here */

    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[ing1]) == -1){ //notifing watcher
    perror ("error on the up operation for semaphore access (AG)");
    exit (EXIT_FAILURE);
    }

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

}
```

- waitForCigarette()

O agente irá esperar que o smoker acabe de enrolar o cigarro. Dentro da zona crítica, muda o estado para WAITING_CIG. Depois, sai da região crítica e faz semDown no semáforo waitCigarette, ou seja, fica à espera de notificação do smoker que acabou de enrolar.

```
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; //changing state
    saveState(nFic, &sh->fSt); //saving state

    /* TODO: insert your code here */

    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) { //waiting for smoker
        perror ("error on the up operation for semaphore access (AG)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
}
```

- closeFactory()

Função feita para ‘fechar a fábrica’ depois de produzidos ingredientes para 5 cigarros. Dentro da zona crítica, muda o estado para CLOSING_A. Muda-se também a variável global ‘closing’ para 1(estado de fecho). Depois, sai da região crítica e faz semUp no semáforo ingredient[id] para todos os ids, ou seja, para notificar os watchers que está a terminar.

```
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; //changing state
    saveState(nFic, &sh->fSt); //saving state

    sh->fSt.closing = 1; //closing is 1

    /* TODO: insert your code here */

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

    for (int index = 0; index < 3; index++){
    if(semUp(semgid, sh->ingredient[index]) == -1){ //notifing watcher
            perror ("error on the up operation for semaphore access (AG)");
            exit (EXIT_FAILURE);
        }
    }
    /* TODO: insert your code here */
}
```

Watcher

Esta entidade funciona como ponte entre o agente e os smokers. Cada watcher comunica com um smoker diferente, e é ele que informa o smoker que pode enrolar.

- waitForIngredient()

A variável ‘ret’ define se a fabrica vai fechar ou não. Tem o valor de ‘false’ se vai fechar e ‘true’ se não. Entrando na região crítica, muda-se o estado para WAITING_ING e volta-se a sair da região crítica. Faz-se semDown do semáforo ingredient para esperar que o agente produza os ingredientes. Também fora da região crítica, verifica-se se a variável global ‘closing’ está a 1 e, se sim, é porque foi colocada pelo agente e indica que vai fechar. Assim, muda-se o estado para CLOSING_W e coloca-se a variável ret a falso(vai fechar). Faz-se semUp do semáforo wait2Ings para informar o smoker.

```
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; //changing state
    saveState(nFic, &sh->fSt); //saveing state
    /* 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){ //wating for ingredient from agent
        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);
    }

    /* 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);
    }

    //closing
    if(sh->fSt.closing){
        sh->fSt.st.watcherStat[id] = CLOSING_W; //changing state
        saveState(nFic, &sh->fSt); //closing state
        ret=false; //should return false if closing
          if  (semUp(semgid,sh->wait2Ings[id]) == -1){ //informing smoker
            perror("error on the up operation for semaphore access (WT)");
            exit (EXIT_FAILURE);
          }
    }

    return ret;

}
```

- updateReservations()

Variável ‘ret’, neste caso, será o id do smoker que tem o ingrediente que falta. Se nenhum tiver, retorna -1. Entra na região critica e altera o estado para UPDATING. De seguida, vai buscar os ingredientes produzidos através da variável global ‘ingredients[id]’ (inventário) e reserva-os (variável global reserved[id]). Depois verifica qual dos smokers tem o ingrediente que falta, retira os ingredientes que vão ser usados dos reservados e retorna o id do smoker que poderá enrolar.

```
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; //changing state
    saveState(nFic, &sh->fSt); //saving state
    int invIng = sh->fSt.ingredients[id]; //invetory of ingredients to reserve
    sh->fSt.reserved[id] = invIng; //reserving ingredient


    int paper = sh->fSt.reserved[HAVEPAPER]; //paper
    int matches = sh->fSt.reserved[HAVEMATCHES]; //matches
    int tabaco = sh->fSt.reserved[HAVETOBACCO]; //tobacco

    //agent produced paper and matches
    if((paper != 0) && (matches != 0)){
    ret = TOBACCO; //smoker that has tobacco may start rolling
    sh->fSt.reserved[HAVEPAPER]--;
    sh->fSt.reserved[HAVEMATCHES]--;
    }
    //agent produced paper and tobacco
    if((paper != 0) && (tabaco != 0)){
    ret = MATCHES; //smoker that has matches may start rolling
    sh->fSt.reserved[HAVEPAPER]--;
    sh->fSt.reserved[HAVETOBACCO]--;
    }
    //agent produced matches and tobacco
    if((matches != 0) && (tabaco != 0)){
    ret = PAPER; //smoker that has paper may start rolling
    sh->fSt.reserved[HAVEMATCHES]--;
    sh->fSt.reserved[HAVETOBACCO]--;
    }


    /* 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;
}
```

- informSmoker()

Entra-se na região crítica e altera-se o estado para INFORMING. Depois, faz-se semUp do semáforo wait2Ings[smokerReady] (smokerReady é a variável passada como argumento que indica o id do smoker que pode enrolar) para notificar o smoker correspondente.

```
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; //changing state
    saveState(nFic, &sh->fSt); //saving state

    if(semUp(semgid, sh->wait2Ings[smokerReady]) == -1){ //notifies smoker with ID=smokerReady to start rolling
        perror ("error on the up operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }

    /* 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);
    }



    /* TODO: insert your code here */
}
```

Smoker

O smoker apenas tem a função de enrolar e fumar o cigarro. Deve também avisar o agente quando acaba de enrolar para este poder começar a produzir novos ingredientes.

- waitForIngredients()

A variável de retorno será 'true' se existem ingredientes disponíveis e falso se vai fechar a fábrica. Entrando na região crítica, muda-se o estado para WAITING_2ING e, saindo, faz-se semDown do semáforo wait2Ings[id], esperando assim pelo sinal do watcher. Depois, voltando a entrar na região crítica, verifica-se se a variável 'closing' tem valor 1 (fábrica a fechar) e, se sim, muda-se o estado para CLOSING_S e coloca-se a variável de retorno a 'false'.

```
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; //changing state
    saveState(nFic, &sh->fSt); //saving state

    /* 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){ //wating for watcher
    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);
    }
    //closing
    if(sh->fSt.closing==1){
    sh->fSt.st.smokerStat[id] = CLOSING_S; //changing state
    saveState(nFic, &sh->fSt); //saving state
    ret = false; //should return false when closing
    }

    /* 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;
}
```

-rollingCigarette()

Aqui é calculado um 'rollingTime', ou seja, será o tempo que o smoker demora a enrolar. Depois, entrando na região crítica, muda-se o estado para ROLLING. Depois, faz-se a atualização do inventário de ingredientes, ou seja, retira-se da variável global 'ingredients' os que o smoker usou para enrolar. Obviamente que dependendo de qual smoker está a enrolar, os ingredientes usados são diferentes. Seguidamente, temos de fazer 'usleep(rollingTime)' para o processo esperar o tempo de enrolar. Antes é verificado se este tempo é superior a 0 para evitar possíveis deadlocks. Já fora da região crítica, faz-se semUp do semáforo waitCigarette para notificar o agente que já pode começar a produzir novos ingredientes.

```
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; //changing state
    saveState(nFic, &sh->fSt); //saving state
    if (id == HAVETOBACCO){
        //smoker one
    sh->fSt.ingredients[HAVEMATCHES]--;
    sh->fSt.ingredients[HAVEPAPER]--;
    } else if(id == HAVEMATCHES){
    //smoker two
    sh->fSt.ingredients[HAVETOBACCO]--;
    sh->fSt.ingredients[HAVEPAPER]--;
    } else{
    //smoker three
    sh->fSt.ingredients[HAVETOBACCO]--;
    sh->fSt.ingredients[HAVEMATCHES]--;
    }
    if(rollingTime > 0){ //avoiding deadlocks
    usleep(rollingTime); //wating rolling time
    }

    /* 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(semUp(semgid, sh->waitCigarette) == -1){ //notifing agent
    perror ("error on the down operation for semaphore access (SM)");
        exit (EXIT_FAILURE);
    }
    /* TODO: insert your code here */

}
```

- smoke()

O valor do tempo que demora a fumar é calculado da mesma maneira que o tempo de enrolar. Entrando na região crítica, muda-se o estado para SMOKING. Incrementa-se também a variável 'nCigarettes[id]', pois vai fumar mais um cigarro. Finalmente, se o 'smokingTime' for positivo, espera o tempo de fumar.

```
static void smoke(int id)
{

    double smokingTime = 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] = SMOKING; //changing state
    saveState(nFic, &sh->fSt); //saving state

    sh->fSt.nCigarettes[id]++; //updating number of cigarettes
    saveState(nFic, &sh->fSt); //saving state
    /* 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(smokingTime > 0){ //avoiding deadlocks
    usleep(smokingTime); //waiting smoking time
    }
    /* TODO: insert your code here */

}
```

Testes e Resultados

Em termos de testes, foram feitos testes a cada uma das entidades indivuais através do '$make ag', '$make wt' e do '$make sm'. Depois, foi feito o teste conjunto com o '$make all'. Este teste conjunto foi corrido várias vezes para se verificar que davam quantidades diferentes de cigarros que cada smoker fumou.

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
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
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
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
1	0	1	0	0	0	0	0	1	1	0	0	0
1	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	0	2	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	1	0	0	0	1	1	0	0	0
2	0	0	0	2	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	0	1	2	0	0	0	1	1	1	0	0
2	0	0	2	2	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	0	0	0	0	1	1	1	0	0
2	0	0	0	1	0	0	0	1	1	1	0	0
2	0	0	0	2	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	0	0	0	0	0	1	1	0	2	0	0
2	1	0	0	0	0	0	1	1	0	2	0	0
2	1	1	0	0	0	0	1	1	0	2	0	0
2	0	1	0	0	0	0	1	1	0	2	0	0
2	0	2	0	0	0	0	1	1	0	2	0	0
2	0	0	0	0	0	0	1	1	0	2	0	0
2	0	0	0	0	0	1	1	1	0	2	0	0
2	0	0	0	0	0	2	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	0	1	2	0	1
2	0	0	0	0	0	0	1	0	1	2	0	1
2	0	0	1	0	0	0	1	0	1	2	0	1
2	1	0	1	0	0	0	1	0	1	2	0	1
2	1	0	0	0	0	0	1	0	1	2	0	1
2	2	0	0	0	0	0	1	0	1	2	0	1
2	0	0	0	0	0	0	1	0	1	2	0	1
2	0	0	0	0	1	0	1	0	1	2	0	1
2	0	0	0	0	2	0	0	0	0	2	0	1
2	0	0	0	0	2	0	0	0	0	2	1	1
1	0	0	0	0	2	0	0	0	0	2	1	1
2	0	0	0	0	2	0	0	1	1	2	1	1
2	0	1	0	0	2	0	0	1	1	2	1	1
2	0	1	1	0	2	0	0	1	1	2	1	1
2	0	1	1	0	0	0	0	1	1	2	1	1
2	0	0	1	0	0	0	0	1	1	2	1	1
2	0	0	2	0	0	0	0	1	1	2	1	1
2	0	0	0	0	0	0	0	1	1	2	1	1
2	0	0	0	1	0	0	0	1	1	2	1	1
2	0	0	0	2	0	0	0	0	0	2	1	1
2	0	0	0	2	0	0	0	0	0	3	1	1
3	0	0	0	2	0	0	0	0	0	3	1	1
3	3	0	0	2	0	0	0	0	0	3	1	1
3	3	3	0	2	0	0	0	0	0	3	1	1
3	3	3	3	2	3	0	0	0	0	3	1	1
3	3	3	3	2	0	0	0	0	0	3	1	1
3	3	3	3	2	3	3	0	0	0	3	1	1
3	3	3	3	0	3	3	0	0	0	3	1	1
3	3	3	3	3	3	3	0	0	0	3	1	1

Conclusão

Com este trabalho foi possível entender de uma forma muito mais clara com funcionam os mecanismos associados à execução e sincronização de processos e threads. Também ajudou a uma melhor compreensão da linguagem de programação C e também de estruturas de dados. Apliquei e percebi os semáforos e como eles funcionam.