



Universidade Federal de Lavras  
PPGCC  
PCC508 – Sistemas Operacionais

## **Tópico 3 Lista Adicional 1**

Douglas Aquino T. Mendes  
21 de outubro de 2024

# Sumário

<b>1</b>	<b>Introdução</b>	<b>2</b>
1.1	Bibliotecas . . . . .	2
<b>2</b>	<b>Atividade 1</b>	<b>2</b>
2.1	Descrição . . . . .	2
2.2	Código . . . . .	3
2.3	Testes e Resultados . . . . .	5
<b>3</b>	<b>Atividade 2</b>	<b>6</b>
3.1	Descrição . . . . .	6
3.2	Código . . . . .	6
3.3	Testes e Resultados . . . . .	7
<b>4</b>	<b>Atividade 3</b>	<b>8</b>
4.1	Descrição . . . . .	8
4.2	Código . . . . .	8
4.3	Testes e Resultados . . . . .	10
<b>5</b>	<b>Atividade 4</b>	<b>10</b>
5.1	Descrição . . . . .	10
5.2	Código . . . . .	10
5.3	Testes e Resultados . . . . .	14
<b>6</b>	<b>Atividade 5</b>	<b>15</b>
6.1	Descrição . . . . .	15
6.2	Código . . . . .	15
6.3	Testes e Resultados . . . . .	17
<b>7</b>	<b>Atividade 6</b>	<b>17</b>
7.1	Descrição . . . . .	17
7.2	Código . . . . .	17
7.3	Testes e Resultados . . . . .	19

# 1 Introdução

Este documento tem como objetivo apresentar o desenvolvimento de atividades adicionais para a disciplina de Sistemas Operacionais, focando na implementação de códigos em linguagem C. Cada atividade foi elaborada com a intenção de abordar conceitos relacionados ao funcionamento de sistemas operacionais, incluindo gerenciamento de processos, gerenciamento de memória, e comunicação entre processos. Serão apresentados os códigos desenvolvidos, seguidos de uma explicação sobre sua lógica de funcionamento e objetivos.

## 1.1 Bibliotecas

Neste trabalho, diversas bibliotecas foram empregadas para facilitar o desenvolvimento dos códigos em linguagem C. Abaixo estão as principais bibliotecas utilizadas, juntamente com suas respectivas funcionalidades:

- **stdio.h**: Fornece funções para entrada e saída padrão, como `printf()` e `scanf()`, permitindo interações com o usuário através do console.
- **stdlib.h**: Contém funções para manipulação de memória dinâmica (`malloc()`, `free()`) e outras utilidades gerais, essenciais para o gerenciamento de recursos.
- **string.h**: Oferece funções para manipulação de strings, como `strlen()`, `strcpy()` e `strcat()`, facilitando o processamento de dados textuais.
- **dirent.h**: Utilizada para manipulação de diretórios, permitindo listar arquivos e acessar informações sobre diretórios no sistema de arquivos.
- **ctype.h**: Fornece funções para testes de caracteres e conversões, como `isdigit()` e `tolower()`, que são úteis para operações relacionadas a caracteres.
- **unistd.h**: Contém declarações para chamadas de sistema POSIX, permitindo operações de baixo nível, como leitura e gravação em arquivos, além de manipulação de processos.
- **sys/types.h**: Define tipos de dados utilizados em chamadas de sistema, incluindo definições para identificadores de processos e modos de arquivos.
- **sys/wait.h**: Oferece macros para controle de processos, permitindo que um processo pai espere a terminação de processos filhos.
- **pthread.h**: Utilizada para programação concorrente, permitindo a criação e manipulação de threads, essencial para o desenvolvimento de aplicações que requerem multitarefa.
- **sys/mman.h**: Contém funções para manipulação de memória, como mapeamento de arquivos e alocação de memória compartilhada.
- **semaphore.h**: Fornece definições e funções para o uso de semáforos, que são utilizados para controlar o acesso a recursos compartilhados em ambientes concorrentes.
- **fcntl.h**: Utilizada para controle de arquivos, permitindo operações como bloqueio de arquivos e manipulação de descritores de arquivos.
- **sys/stat.h**: Define a estrutura de dados e funções para obter informações sobre arquivos, como tipo, permissões e tamanhos.

## 2 Atividade 1

### 2.1 Descrição

O diretório `/proc/PID` apresenta vários arquivos com informações sobre os processos em execução na máquina. Em especial, o arquivo `status` apresenta informações de estado sobre um processo. Fazer um programa que receba uma string na entrada e apresente as informações do arquivo `status` de todos os processos que tenham essa string como substring do nome do processo.

## 2.2 Código

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include <string.h>
5 #include <dirent.h>
6 #include <ctype.h>
7
8 int is_numeric(const char *str)
9 {
10     for (int i = 0; str[i] != '\0'; i++)
11     {
12         if (!isdigit(str[i]))
13         {
14             return 0;
15         }
16     }
17     return 1;
18 }
19
20 void print_process_status(const char *pid)
21 {
22     char status_path[256];
23     snprintf(status_path, sizeof(status_path), "/proc"
24              "%s"
25              "/status",
26              pid);
27
28     FILE *status_file = fopen(status_path, "r");
29     if (!status_file)
30     {
31         return;
32     }
33
34     char line[256];
35     while (fgets(line, sizeof(line), status_file))
36     {
37         printf("%s", line);
38     }
39
40     fclose(status_file);
41 }
42
43 int main()
44 {
45     char process_name[256];
46
47     printf("Digite a string de busca: ");
48     scanf("%s", process_name);
49
50     DIR *proc_dir = opendir("/proc");
51
52     if (!proc_dir)
```

```

53 {
54     perror("Erro ao abrir o diretorio /proc");
55     return 1;
56 }
57
58 struct dirent *entry;
59 while ((entry = readdir(proc_dir)) != NULL)
60 {
61     if (is_numeric(entry->d_name))
62     {
63         char status_path[256];
64         snprintf(status_path, sizeof(status_path), "/proc"
65                 "/%s"
66                 "/status",
67                 entry->d_name);
68
69         FILE *status_file = fopen(status_path, "r");
70
71         if (!status_file)
72         {
73             continue;
74         }
75
76         char line[256];
77         int found = 0;
78         while (fgets(line, sizeof(line), status_file))
79         {
80             if (strncmp(line, "Name:", strlen("Name:")) == 0)
81             {
82                 char *name = line + strlen("Name:") + 1;
83                 if (strstr(name, process_name) != NULL)
84                 {
85                     printf("Processo com PID %s encontrado:\n", entry->
86                             d_name);
87                     print_process_status(entry->d_name);
88                     printf("\n");
89                     found = 1;
90                 }
91                 break;
92             }
93
94             fclose(status_file);
95
96             if (!found)
97             {
98                 continue;
99             }
100         }
101     }
102
103     closedir(proc_dir);
104     return 0;
105 }

```

## 2.3 Testes e Resultados

Como teste, foi dado a string "bash" como entrada como é ilustrado na figura 1, e alguns dos resultados podem ser visualizados nas figuras 2 e 3.

```
Codes$ ./atv1
Digite a string de busca: bash
```

Figura 1: String de entrada

```
Codes$ ./atv1
Digite a string de busca: bash
Processo com PID 5180 encontrado:
Name: bash
Umask: 0002
State: S (sleeping)
Tgid: 5180
Ngid: 0
Pid: 5180
PPid: 5092
TracerPid: 0
Uid: 1000 1000 1000 1000
Gid: 1000 1000 1000 1000
FDSize: 256
Groups: 4 20 24 27 30 46 122 134 135 1000
NSTgid: 5180
NSpid: 5180
NSpgid: 5180
NSSid: 5180
Kthread: 0
VmPeak: 11572 kB
VmSize: 11572 kB
```

Figura 2: Resultado 1 da execução do programa

```

x86_Thread_features_locked:

Processo com PID 7274 encontrado:
Name:  bash
Umask: 0002
State: S (sleeping)
Tgid:  7274
Ngid:  0
Pid:   7274
PPid:  7256
TracerPid: 0
Uid:   1000    1000    1000    1000
Gid:   1000    1000    1000    1000
FDSize: 256
Groups: 4 20 24 27 30 46 122 134 135 1000
NSTgid: 7274
NSpid:  7274
NSpgid: 7274
NSSid:  7274
Kthread: 0
VmPeak:  11544 kB
VmSize:  11512 kB

```

Figura 3: Resultado 2 da execução do programa

## 3 Atividade 2

### 3.1 Descrição

Um shell no Unix é responsável por receber comandos do usuário e executá-los, em linha de comando. Criar um mini-shell que permita o usuário executar comandos que são executáveis no Linux. Para isso, use as chamadas de sistema fork/execve. Permita também o usuário listar os arquivos do diretório atual com ls, que nesse caso será implementado como comando interno.

### 3.2 Código

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include <string.h>
5 #include <unistd.h>
6
7 #include <sys/types.h>
8 #include <sys/wait.h>
9
10 #define MAX_CMD_LEN 1024
11 #define MAX_ARGS 100
12
13 int main()
14 {
15     char command[MAX_CMD_LEN];
16
17     while (1)
18     {
19
20         printf(" mini-shell> ");

```

```

21     fflush(stdout);
22
23     if (fgets(command, sizeof(command), stdin) == NULL)
24     {
25         perror("fgets");
26         continue;
27     }
28
29     command[strcspn(command, "\n")] = '\0';
30
31     if (strcmp(command, "exit") == 0)
32     {
33         break;
34     }
35
36     if (fork() == 0)
37     { // Processo filho
38         char *args[MAX_ARGS];
39         char *token = strtok(command, " ");
40         int i;
41
42         for (i = 0; token != NULL && i < MAX_ARGS - 1; i++)
43         {
44             args[i] = token;
45             token = strtok(NULL, " ");
46         }
47         printf("i: %d\n", i);
48         args[i] = NULL;
49
50         execvp(args[0], args);
51         perror("execvp");
52         exit(EXIT_FAILURE);
53     }
54     else
55     { // Processo pai
56         wait(NULL); // Espera o filho terminar
57     }
58 }
59
60 return 0;
61 }

```

### 3.3 Testes e Resultados

Para teste da atividade 2, primeiramente tentamos o comando ls que lista o diretório atual, como é possível observar na figura 4, o diretório apenas possui os arquivos das atividades. Infelizmente o programa desenvolvido não colore os arquivos de acordo com sua propriedade, com isto, fica difícil diferenciar os arquivos de texto, de executáveis e pastas. Continuando os testes, foi criado um novo diretório chamado "teste" e novamente ls para mostrar que agora há um novo diretório. Como dificuldade, enquanto estava desenvolvendo o programa, por conta de um erro de lógica, o argumento não era passado corretamente, e acabei criando uma pasta com alguma referencia de lixo de memória a qual não conseguia excluir, como pode ser visto na figura 5



```

Codes$ ./atv2
mini-shell> ls
i: 1
atv1 atv2 atv3 atv4 atv5 atv6
atv1.c atv2.c atv3.c atv4.c atv5.c atv6.c
mini-shell> mkdir teste
i: 2
mini-shell> ls
i: 1
atv1 atv2 atv3 atv4 atv5 atv6 teste
atv1.c atv2.c atv3.c atv4.c atv5.c atv6.c
mini-shell> exit
douglass@Virtuoso:~/Documentos/Projetos/Disciplinas/Mestrado_S0/
Codes$

```

Figura 4: Resultado da execução do programa 2

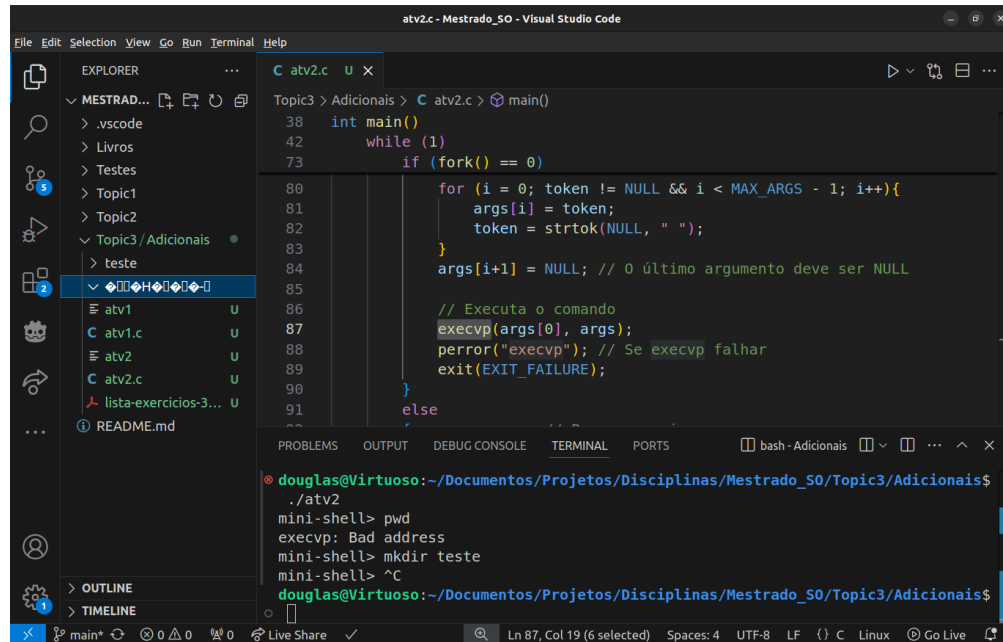


Figura 5: Resultado com erro da execução do programa 2

## 4 Atividade 3

### 4.1 Descrição

A alternância estrita é um método de obtenção de exclusão mútua com espera ocupada. Implementar 2 threads que incrementam um contador comum, onde o controle de concorrência é feito utilizando alternância estrita.

### 4.2 Código

```

1 /*
2 gcc -o atv3 atv3.c -lpthread
3 ./atv3
4 */
5
6 #include <stdio.h>
7 #include <stdlib.h>
8
9 #include <pthread.h>

```

```

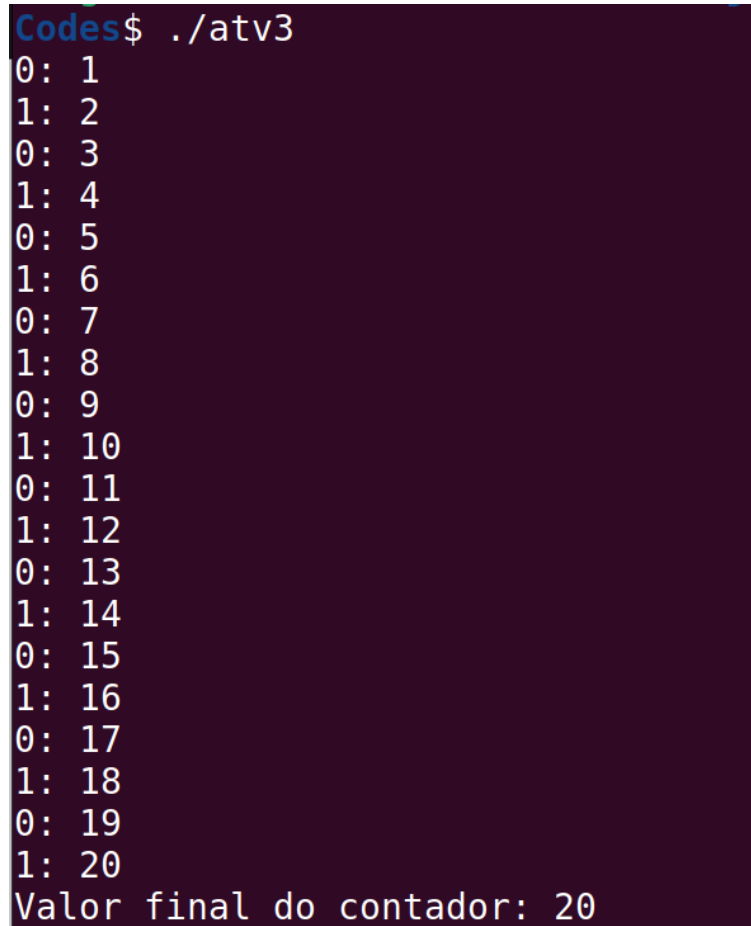
10 #include <unistd.h>
11
12 volatile int contador = 0;
13 pthread_mutex_t mutex;
14
15 volatile int turno;
16
17 void *incrementa_contador(void *arg)
18 {
19     int id = *(int *)arg;
20
21     for (int i = 0; i < 10; i++)
22     {
23         while (1)
24         {
25             pthread_mutex_lock(&mutex);
26             if (turno == id)
27             {
28                 contador++;
29                 printf("%d: %d\n", id, contador);
30                 turno = (turno + 1) % 2;
31                 pthread_mutex_unlock(&mutex);
32                 break;
33             }
34             pthread_mutex_unlock(&mutex);
35         }
36     }
37
38     return NULL;
39 }
40
41 int main()
42 {
43     pthread_t threads[2];
44     pthread_mutex_init(&mutex, NULL);
45     int ids[2] = {0, 1};
46
47     for (int i = 0; i < 2; i++)
48     {
49         if (pthread_create(&threads[i], NULL, incrementa_contador, &ids[i]) !=
50             0)
51         {
52             perror("Falha ao criar thread");
53             return 1;
54         }
55     }
56
57     for (int i = 0; i < 2; i++)
58     {
59         pthread_join(threads[i], NULL);
60     }
61
62     pthread_mutex_destroy(&mutex);

```

```
63     return 0;  
64 }
```

### 4.3 Testes e Resultados

Como é possível observar no código fonte do programa, o printf mostra no terminal qual a thread está incrementando e o valor o qual ficou o contador, com isto, temos o resultado na figura 6.



```
Codes$ ./atv3  
0: 1  
1: 2  
0: 3  
1: 4  
0: 5  
1: 6  
0: 7  
1: 8  
0: 9  
1: 10  
0: 11  
1: 12  
0: 13  
1: 14  
0: 15  
1: 16  
0: 17  
1: 18  
0: 19  
1: 20  
Valor final do contador: 20
```

Figura 6: Resultado da execução do programa 3

## 5 Atividade 4

### 5.1 Descrição

Implementar o problema dos filósofos gluttons com semáforos nomeados. Cada filósofo será implementado como um processo. Para o controle de concorrência, serão utilizados uma série de semáforos nomeados. O programa deverá receber um parâmetro que é o número de filósofos a serem criados. Para criação de processos, a chamada fork deve ser utilizada.

### 5.2 Código

```

1 #define _GNU_SOURCE
2
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 #include <sys/wait.h>
7 #include <sys/types.h>
8 #include <unistd.h>
9 #include <sys/mman.h>
10 #include <semaphore.h>
11 #include <fcntl.h>
12 #include <sys/stat.h>
13
14 #define PENSANDO 0
15 #define COMFOME 1
16 #define COMENDO 2
17
18 int qnt_filosofos;
19
20 int *state;
21 sem_t *mutex, *semaf;
22
23 void printStates()
24 {
25     printf("Estados: ");
26     for (int i = 0; i < qnt_filosofos; i++)
27     {
28         printf("%d ", state[i]);
29     }
30     printf("\n");
31 }
32
33 void teste(int filosofo)
34 {
35     printf("Filosofo %d verificando se pode comer\n", filosofo);
36
37     /*printf("%d %d %d\n", state[filosofo] == COMFOME,
38         state[(filosofo + qnt_filosofos - 1) % qnt_filosofos] != COMENDO,
39         state[(filosofo + 1) % qnt_filosofos] != COMENDO);*/
40
41     sem_wait(mutex);
42     if (state[filosofo] == COMFOME &&
43         state[(filosofo + qnt_filosofos - 1) % qnt_filosofos] != COMENDO &&
44         state[(filosofo + 1) % qnt_filosofos] != COMENDO)
45     {
46
47         printf("O %d pode!\n", filosofo);
48         // printStates();
49         state[filosofo] = COMENDO;
50         // printStates();
51         sem_post(&semaf[filosofo]);
52     }
53     sem_post(mutex);

```

```

54 }
55
56 void pensando(int filosofo)
57 {
58     printf("Filosofo %d pensando.\n", filosofo);
59     sleep(rand() % 3 + 1);
60 }
61
62 void comendo(int filosofo)
63 {
64     printf("Filosofo %d comendo.\n", filosofo);
65     sleep(rand() % 3 + 1);
66 }
67
68 void pegandoGarfo(int filosofo)
69 {
70     printf("Filosofo %d preparando para pegar os garfos\n", filosofo);
71     sem_wait(mutex);
72
73     //printf("O estado do filosofo %d mudou DE: %d\n", filosofo, state[
74         filosofo]);
75     // printStates();
76     state[filosofo] = COMFOME;
77     // printStates();
78     //printf("%d (1)PARA: %d\n", filosofo, state[filosofo]);
79     sem_post(mutex);
80
81     teste(filosofo);
82
83     sem_wait(&semaf[filosofo]);
84     printf("filosofo %d pegou os garfos!\n", filosofo);
85 }
86
87 void largandoGarfo(int filosofo)
88 {
89     printf("Filosofo %d largando os garfos\n", filosofo);
90     sem_wait(mutex);
91
92     //printf("O estado do filosofo %d mudou DE: %d\n", filosofo, state[
93         filosofo]);
94     // printStates();
95     state[filosofo] = PENSANDO;
96     // printStates();
97     //printf("%d (0)PARA: %d\n", filosofo, state[filosofo]);
98
99     //printf("Esquedo: %d Direito: %d \n", ((filosofo + qnt_filosofos - 1) %
100         qnt_filosofos), ((filosofo + 1) % qnt_filosofos));
101     sem_post(mutex);
102
103     teste((filosofo + qnt_filosofos - 1) % qnt_filosofos);
104     teste((filosofo + 1) % qnt_filosofos);
105 }
106
107 void rotinaDoFilosofo(int filosofo)

```

```

105 {
106     while (1)
107     {
108         pensando(filosofo);
109         pegandoGarfo(filosofo);
110         comendo(filosofo);
111         largandoGarfo(filosofo);
112     }
113 }
114
115 int main()
116 {
117
118     printf("Qual a quantidade de filosofos?:\n");
119     scanf("%d", &qnt_filosofos);
120
121     state = mmap(NULL, (sizeof(int) * qnt_filosofos), PROT_READ | PROT_WRITE,
122                  MAP_SHARED | MAP_ANONYMOUS, -1, 0);
123
124     mutex = mmap(NULL, (sizeof(sem_t)), PROT_READ | PROT_WRITE, MAP_SHARED |
125                  MAP_ANONYMOUS, -1, 0);
126     sem_init(mutex, 1, 1);
127
128     semaf = mmap(NULL, (sizeof(sem_t) * qnt_filosofos), PROT_READ | PROT_WRITE
129                  , MAP_SHARED | MAP_ANONYMOUS, -1, 0);
130
131     for (int i = 0; i < qnt_filosofos; i++)
132     {
133         sem_init(&semaf[i], 1, 0);
134     }
135
136     for (int i = 0; i < qnt_filosofos; i++)
137     {
138         if (fork() == 0)
139         {
140             rotinaDoFilosofo(i);
141         }
142     }
143
144     while (1)
145     {
146         sleep(1);
147     }
148
149     munmap(state, sizeof(int) * qnt_filosofos);
150     munmap(mutex, sizeof(sem_t));
151     munmap(semaf, sizeof(sem_t) * qnt_filosofos);
152
153     return 0;
154 }

```

### 5.3 Testes e Resultados

A figura 7 irá colaborar para compreensão e conferência dos resultados obtidos na execução do programa, que pode ser observado nas figura 8 e 9.

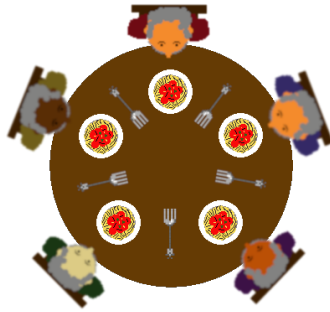


Figura 7: Representação dos filósofos em uma mesa

```
Codes$ ./atv4
Qual a quantidade de filósofos?:
5
Filosofo 0 pensando.
Filosofo 1 pensando.
Filosofo 2 pensando.
Filosofo 3 pensando.
Filosofo 4 pensando.
Filosofo 2 preparando para pegar os garfos
Filosofo 0 preparando para pegar os garfos
Filosofo 1 preparando para pegar os garfos
Filosofo 3 preparando para pegar os garfos
Filosofo 2 verificando se pode comer
Filosofo 3 verificando se pode comer
0 3 pode!
Filosofo 0 verificando se pode comer
0 0 pode!
filosofo 3 pegou os garfos!
Filosofo 3 comendo.
filosofo 0 pegou os garfos!
Filosofo 0 comendo.
Filosofo 1 verificando se pode comer
Filosofo 4 preparando para pegar os garfos
```

Figura 8: Exucução do programa 4

```
Filosofo 1 verificando se pode comer
Filosofo 4 preparando para pegar os garfos
Filosofo 4 verificando se pode comer
Filosofo 0 largando os garfos
Filosofo 3 largando os garfos
Filosofo 4 verificando se pode comer
Filosofo 2 verificando se pode comer
0 4 pode!
Filosofo 1 verificando se pode comer
0 2 pode!
Filosofo 4 verificando se pode comer
filosofo 4 pegou os garfos!
Filosofo 0 pensando.
Filosofo 3 pensando.
Filosofo 4 comendo.
filosofo 2 pegou os garfos!
Filosofo 2 comendo.
Filosofo 3 preparando para pegar os garfos
Filosofo 0 preparando para pegar os garfos
Filosofo 3 verificando se pode comer
Filosofo 0 verificando se pode comer
Filosofo 4 largando os garfos
Filosofo 2 largando os garfos
```

Figura 9: Continuação da execução do programa

## 6 Atividade 5

### 6.1 Descrição

Implementar o problema dos produtores/consumidores utilizando threads e mutexes, juntamente com variáveis de condição.

### 6.2 Código

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include <pthread.h>
5 #include <unistd.h>
6
7 #define BUFFER_SIZE 5
8 #define NUMITEMS 20
9
10 int buffer[BUFFER_SIZE];
11 int in = 0;
12 int out = 0;
13 pthread_mutex_t mutex;
14 pthread_cond_t not_full;
15 pthread_cond_t not_empty;
16
17 void *producer(void *arg)
18 {
19     for (int i = 0; i < NUMITEMS; i++)
20     {
21         int item = rand() % 100;
22
23         pthread_mutex_lock(&mutex);
24
25         while ((in + 1) % BUFFER_SIZE == out)
26         {
27             pthread_cond_wait(&not_full, &mutex);
28         }
29
30         buffer[in] = item;
31         printf("Produzido: %d\n", item);
32         in = (in + 1) % BUFFER_SIZE;
33
34         pthread_cond_signal(&not_empty);
35         pthread_mutex_unlock(&mutex);
36
37         sleep(rand() % 2);
38     }
39     //printf("Fim de producao\n");
40     return NULL;
41 }
42
43 void *consumer(void *arg)
44 {
45     int items_consumed;
```



```

46
47     for (int i = 0; i < NUMITEMS; i++)
48     {
49         pthread_mutex_lock(&mutex);
50
51         while (in == out)
52         {
53             pthread_cond_wait(&not_empty, &mutex);
54         }
55
56         items_consumed = 0;
57         while (in != out)
58         {
59             int item = buffer[out];
60             printf("Consumido: %d\n", item);
61             out = (out + 1) % BUFFER_SIZE;
62             items_consumed++;
63         }
64
65         printf("Total consumido nessa rodada: %d\n", items_consumed);
66
67         pthread_cond_signal(&not_full);
68         pthread_mutex_unlock(&mutex);
69
70         sleep(rand() % 5);
71     }
72     printf("Fim de consumo\n");
73     return NULL;
74 }
75
76 int main()
77 {
78     pthread_t prod_thread, cons_thread;
79
80     pthread_mutex_init(&mutex, NULL);
81     pthread_cond_init(&not_full, NULL);
82     pthread_cond_init(&not_empty, NULL);
83
84     pthread_create(&prod_thread, NULL, producer, NULL);
85     pthread_create(&cons_thread, NULL, consumer, NULL);
86
87     pthread_join(prod_thread, NULL);
88     pthread_join(cons_thread, NULL);
89
90     pthread_mutex_destroy(&mutex);
91     pthread_cond_destroy(&not_full);
92     pthread_cond_destroy(&not_empty);
93
94     return 0;
95 }

```

```

douglas@Virtuoso:~/Documentos/Projetos/Disci
Codes$ ./atv5
Produzido: 83
Consumido: 83
Total consumido nessa rodada: 1
Produzido: 15
Produzido: 35
Produzido: 92
Consumido: 15
Consumido: 35
Consumido: 92
Total consumido nessa rodada: 3
Produzido: 62
Consumido: 62
Total consumido nessa rodada: 1
Produzido: 59
Consumido: 59
Total consumido nessa rodada: 1
Produzido: 40
Consumido: 40
Total consumido nessa rodada: 1
Produzido: 36
Produzido: 68

```

Figura 10: Resultado da execução do programa 5

### 6.3 Testes e Resultados

## 7 Atividade 6

### 7.1 Descrição

Implementar o problema dos leitores/escritores com um método de IPC a escolha.

### 7.2 Código

```

1 #define _GNU_SOURCE
2
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 #include <sys/wait.h>
7 #include <sys/types.h>
8 #include <unistd.h>
9 #include <sys/mman.h>
10 #include <semaphore.h>
11 #include <fcntl.h>
12 #include <sys/stat.h>
13
14 int qnt_Leitores, qnt_Escritores, *qnt_Lendo;
15 sem_t *mutex_Lendo, *semaf_Escrita;
16
17 void Escrever(int escritor)

```

```

18 {
19     for (int i = 0; i < 5; i++)
20     {
21         printf("Escritor %d quer escrever . . . \n\n", escritor);
22
23         sem_wait(semaf_Escrita);
24
25         printf("Escritor %d escrevendo . . . \n\n", escritor);
26         sleep(rand() % 3 + 1);
27
28         sem_post(semaf_Escrita);
29         sleep(rand() % 3 + 1);
30     }
31 }
32
33 void Ler(int leitor)
34 {
35     for (int i = 0; i < 5; i++)
36     {
37         printf("Leitor %d se preparando para ler . . . \n\n", leitor);
38
39         sem_wait(mutex_Lendo);
40
41         (*qnt_Lendo)++;
42         printf("Leitores lendo no momento: %d \n\n", *qnt_Lendo);
43
44         if (*qnt_Lendo == 1)
45         {
46             sem_wait(semaf_Escrita);
47         }
48
49         sem_post(mutex_Lendo);
50         printf("Leitor %d lendo . . . \n\n", leitor);
51         sleep(rand() % 3 + 1);
52
53         sem_wait(mutex_Lendo);
54         (*qnt_Lendo)--;
55
56         if (*qnt_Lendo == 0)
57         {
58             sem_post(semaf_Escrita);
59         }
60         sem_post(mutex_Lendo);
61     }
62 }
63
64 int main()
65 {
66
67     printf("Qual a quantidade de Leitores?:\n");
68     scanf("%d", &qnt_Leitores);
69
70     printf("Qual a quantidade de Escritores?:\n");
71     scanf("%d", &qnt_Escritores);

```

```

72
73 qnt_Lendo = mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED |
74 MAP_ANONYMOUS, -1, 0);
75 *qnt_Lendo = 0;
76
77 mutex_Lendo = mmap(NULL, (sizeof(sem_t)), PROT_READ | PROT_WRITE,
78 MAP_SHARED | MAP_ANONYMOUS, -1, 0);
79 sem_init(&mutex_Lendo, 1, 1);
80
81 semaf_Escrita = mmap(NULL, (sizeof(sem_t)), PROT_READ | PROT_WRITE,
82 MAP_SHARED | MAP_ANONYMOUS, -1, 0);
83 sem_init(&semaf_Escrita, 1, 1);
84
85 for (int i = 0; i < qnt_Leitores; i++)
86 {
87     if (fork() == 0)
88     {
89         Ler(i);
90     }
91     else
92     {
93         // printf("Pai\n");
94     }
95 }
96 for (int i = 0; i < qnt_Escritores; i++)
97 {
98     if (fork() == 0)
99     {
100         Escrever(i);
101     }
102     else
103     {
104         // printf("Pai\n");
105     }
106 }
107
108 for (int i = 0; i < qnt_Leitores + qnt_Escritores; i++)
109 {
110     wait(NULL);
111 }
112
113 sem_destroy(&mutex_Lendo);
114 sem_destroy(&semaf_Escrita);
115 munmap(qnt_Lendo, sizeof(int));
116 munmap(&mutex_Lendo, sizeof(sem_t));
117 munmap(&semaf_Escrita, sizeof(sem_t));
118
119 return 0;
120 }

```

### 7.3 Testes e Resultados

```
Escritor 0 quer escrever . . .  
Escritor 1 quer escrever . . .  
Escritor 0 está escrevendo . . .  
Leitor 4 está lendo . . .  
Leitores lendo no momento: 2  
Leitor 3 está lendo . . .  
Leitores lendo no momento: 3  
Leitor 2 está lendo . . .  
Leitores lendo no momento: 4  
Leitor 4 está lendo . . .  
Leitores lendo no momento: 5  
Leitor 1 está lendo . . .
```

Figura 11: Resultado da execução do programa 6