PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Instituto de Ciências Exatas e de Informática

Saulo de Moura Zandona Freitas[1]

Lista #7

Computação Distribuída

[1]Aluno de Graduação em Ciência da Computação– saulomzf@gmail.com

Na nossa página inicial do canvas, há um módulo chamado SMPL. com arquivos da biblioteca de simulação de eventos discretos, o smpl. Faça as 4 subtarefas a seguir, poste prints dos códigos e das execuções em um arquivo em branco, convertido em pdf:

Tarefa 7.0: digitar, compilar e executar o programa exemplo, tempo.c

```c
/* tempo0.c */

#include <stdio.h>
#include <stdlib.h>
#include "smpl.h"

#define test 1
#define fault 2
#define recovery 3

typedef struct {
   int id;
} TipoProcesso;

TipoProcesso *processo;

int main(int argc, char *argv[]) {
    static int N, token, event, r, i;
    static char fa_name[5];

    if (argc != 2) {
        puts("Uso correto: tempo <num-processos>");
        exit(1);
    }

    N = atoi(argv[1]);
    smpl(0, "Um Exemplo de Simula  o");

    processo = (TipoProcesso *) malloc(N * sizeof(TipoProcesso));
    for (i = 0; i < N; i++) {
        sprintf(fa_name, "%d", i);
        processo[i].id = facility(fa_name, 1);
```

```
33      }
34
35      for (i = 0; i < N; i++) {
36          schedule(test, 30.0, i);
37      }
38
39      while (time() < 100.0) {
40          cause(&event, &token);
41          switch (event) {
42              case test:
43                  printf("O processo %d est  funcionando no tempo %g\n",
44                         token, time());
45                  schedule(test, 30.0, token);
46                  break;
47          }
48      }
49
50      free(processo);
51      return 0;
52  }
```

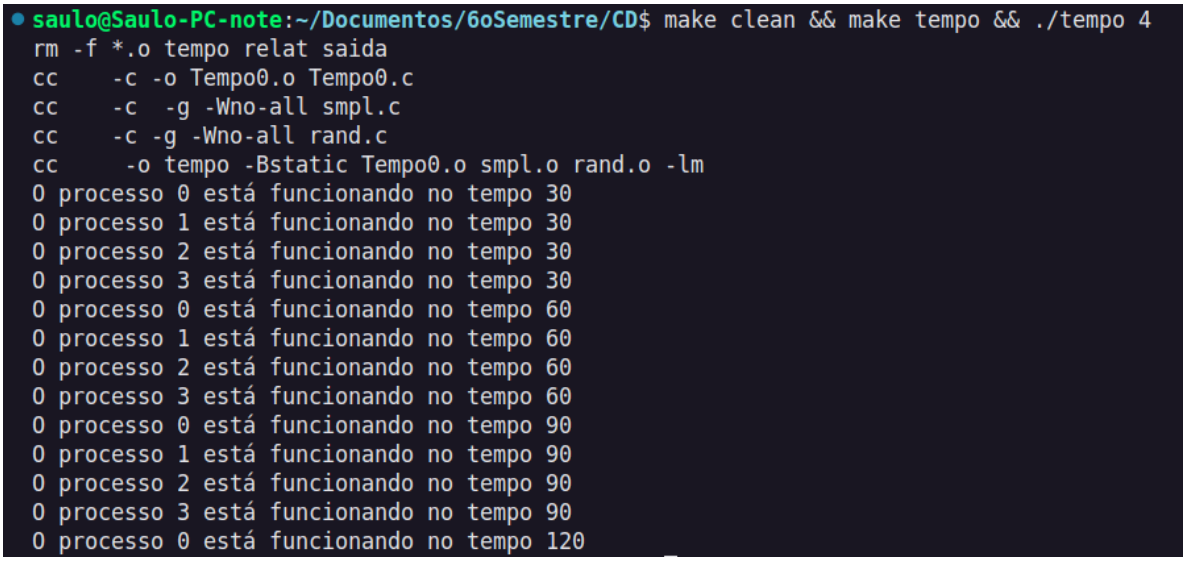

Figura 1: Print do terminal com a execução do algoritmo da subtarefa 7.0

Tarefa 7.1: Fazer cada um dos processos testar o seguinte no anel. Implemente o teste com a função status() do SMPL e imprimir (printf) o resultado de cada teste executado. Por exemplo: "O processo i testou o processo j correto no tempo tal."

```
1  /* tempo1.c */
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include "smpl.h"
6
7  #define test 1
8
9  typedef struct {
10     int id;
11 } TipoProcesso;
12
13 TipoProcesso *processo;
14
15 int main(int argc, char *argv[]) {
16     static int N, token, event, r, i, next;
17     static char fa_name[5];
18
19     if (argc != 2) {
20         puts("Uso correto: tempo <num-processos>");
21         exit(1);
22     }
23
24     N = atoi(argv[1]);
25     smpl(0, "Simula  o de Testes");
26
27     processo = (TipoProcesso *) malloc(N * sizeof(TipoProcesso));
28     for (i = 0; i < N; i++) {
29         sprintf(fa_name, "%d", i);
30         processo[i].id = facility(fa_name, 1);
31     }
32
33     for (i = 0; i < N; i++) {
34         schedule(test, 30.0, i);
35     }
36
```

```
37      while (time() < 100.0) {
38          cause(&event, &token);
39          switch (event) {
40              case test:
41                  next = (token + 1) % N;
42                  r = status(processo[next].id);
43                  if (r == 0)
44                      printf("O processo %d testou o processo %d
45                      correto no tempo %g\n", token, next, time());
46                  else
47                      printf("O processo %d testou o processo %d
48                      falho no tempo %g\n", token, next, time());
49                  schedule(test, 30.0, token);
50                  break;
51          }
52      }
53
54      free(processo);
55      return 0;
56  }
57
58  }
```

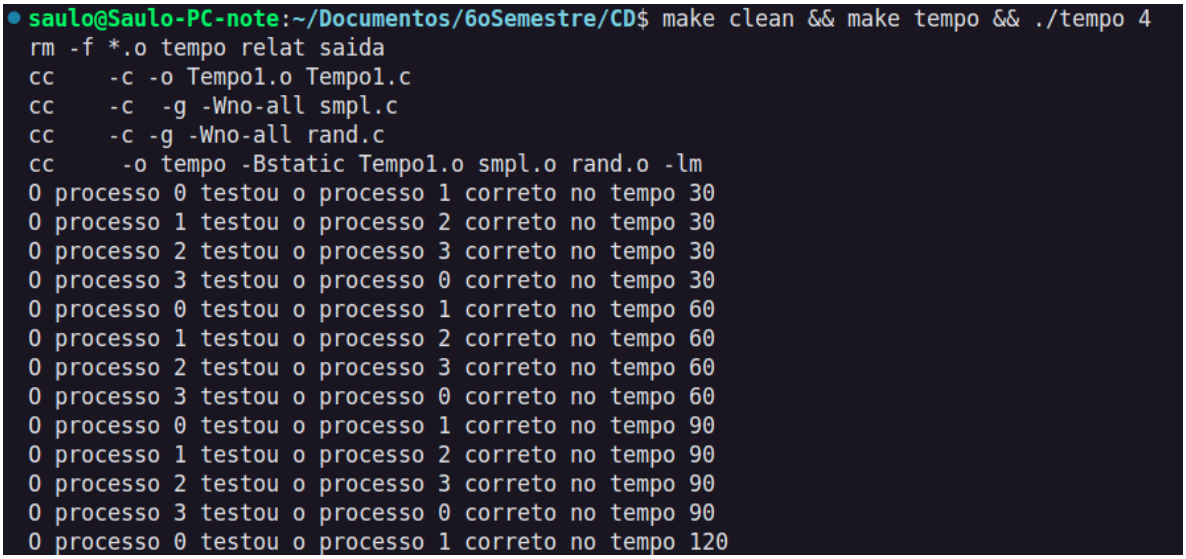

Figura 2: Print do terminal com a execução do algoritmo da subtarefa 7.1

Tarefa 7.2: Cada processo correto executa testes até achar outro processo correto. Lembre-se de tratar o caso em que todos os demais processos estão falhos. Imprimir os testes e resultados.

```
1  /* tempo2 */
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include "smpl.h"
6
7  #define test 1
8
9  typedef struct {
10    int id;
11 } TipoProcesso;
12
13 TipoProcesso *processo;
14
15 int main(int argc, char *argv[]) {
16     static int N, token, event, r, i, next, found;
17     static char fa_name[5];
18
19     if (argc != 2) {
20         puts("Uso correto: tempo <num-processos>");
21         exit(1);
22     }
23
24     N = atoi(argv[1]);
25     smpl(0, "Simula  o de Testes at  Encontrar Corretos");
26
27     processo = (TipoProcesso *) malloc(N * sizeof(TipoProcesso));
28     for (i = 0; i < N; i++) {
29         sprintf(fa_name, "%d", i);
30         processo[i].id = facility(fa_name, 1);
31     }
32
33     for (i = 0; i < N; i++) {
34         schedule(test, 30.0, i);
35     }
36
37     while (time() < 100.0) {
```

```
38          cause(&event, &token);
39          switch (event) {
40              case test:
41                  found = 0;
42                  for (i = 1; i < N && !found; i++) {
43                      next = (token + i) % N;
44                      r = status(processo[next].id);
45                      if (r == 0) {
46                          printf("O processo %d encontrou
47                          o processo %d correto no tempo %g\n",
48                          token, next, time());
49                          found = 1;
50                      } else {
51                          printf("O processo %d testou o processo %d
52                          falho no tempo %g\n", token, next, time());
53                      }
54                  }
55                  if (!found)
56                      printf("O processo %d n o encontrou nenhum
57                      processo correto.\n", token);
58                  schedule(test, 30.0, token);
59                  break;
60          }
61      }
62
63      free(processo);
64      return 0;
65  }
```

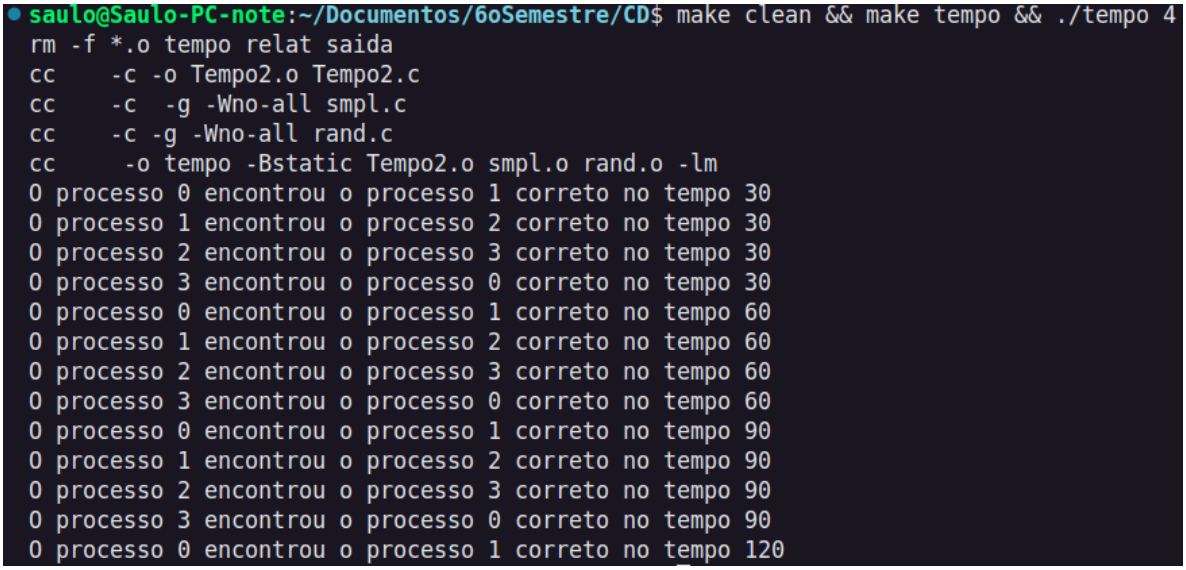

Figura 3: Print do terminal com a execução do algoritmo da subtarefa 7.2

Tarefa 7.3: Cada processo mantém localmente o vetor State[N]. A entrada do vetor State[j] indica o estado do processo j. O estado de cada processo pode ser: -1 (unknown?), 0 (correto) ou 1 (falho). Inicialize (para todos os processos) o State[N] com -1 (indicando estado "unknown") para todos os demais processos e 0 para o próprio processo. Nesta tarefa ao executar um teste em um processo j, o testador atualiza a entrada correspondente no vetor State[j]. Em cada intervalo de testes, mostre o vetor State[N].

```c
/* tempo_task_7_3.c */

#include <stdio.h>
#include <stdlib.h>
#include "smpl.h"

#define test 1
#define UNKNOWN -1
#define CORRECT 0
#define FAULTY 1

typedef struct {
   int id;
   int state;
} TipoProcesso;

TipoProcesso *processo;
int State[10]; // Ajustar tamanho conforme o n mero de processos

int main(int argc, char *argv[]) {
    static int N, token, event, r, i, next;
    static char fa_name[5];

    if (argc != 2) {
        puts("Uso correto: tempo <num-processos>");
        exit(1);
    }

    N = atoi(argv[1]);
    smpl(0, "Simula  o com Vetor de Estados");

    processo = (TipoProcesso *) malloc(N * sizeof(TipoProcesso));
```

```
33      for (i = 0; i < N; i++) {
34          sprintf(fa_name, "%d", i);
35          processo[i].id = facility(fa_name, 1);
36          processo[i].state = CORRECT;
37          for (int j = 0; j < N; j++) {
38              State[j] = (j == i) ? CORRECT : UNKNOWN;
39          }
40      }
41
42      for (i = 0; i < N; i++) {
43          schedule(test, 30.0, i);
44      }
45
46      while (time() < 100.0) {
47          cause(&event, &token);
48          switch (event) {
49              case test:
50                  next = (token + 1) % N;
51                  r = status(processo[next].id);
52                  State[next] = (r == 0) ? CORRECT : FAULTY;
53                  printf("O processo %d atualizou o estado
54                  do processo %d para %d no tempo %g\n",
55                      token, next, State[next], time());
56
57                  printf("Vetor de Estados do processo %d: ", token);
58                  for (int j = 0; j < N; j++) {
59                      printf("%d ", State[j]);
60                  }
61                  printf("\n");
62
63                  schedule(test, 30.0, token);
64                  break;
65          }
66      }
67
68      free(processo);
69      return 0;
70  }
```

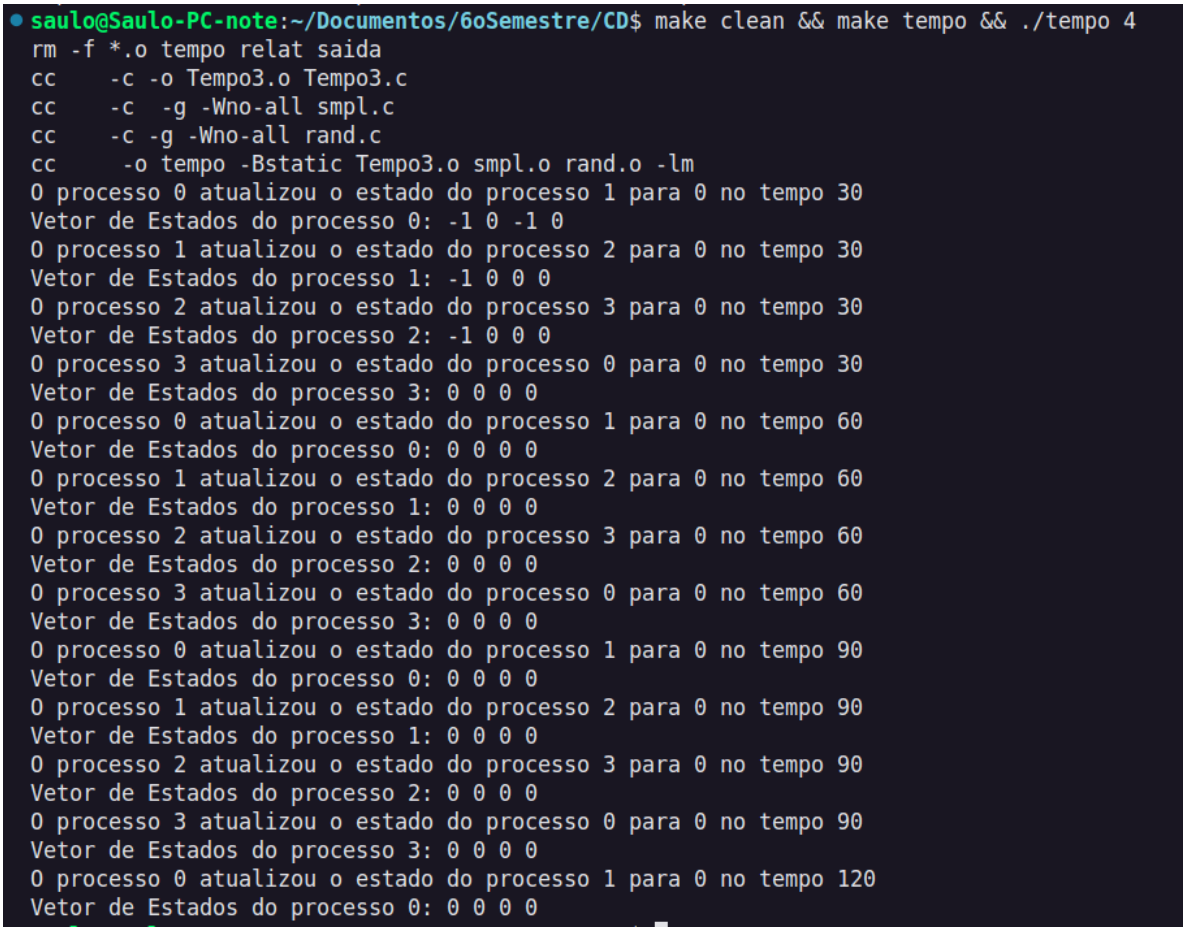

Figura 4: Print do terminal com a execução do algoritmo da subtarefa 7.3

Tarefa 7.4: Quando um processo correto testa outro processo correto, obtém as informações do estado dos demais processos do sistema, processos do sistema exceto aqueles que testou nesta rodada, além do próprio testador.

```
1  /* tempo_task_7_4.c */
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include "smpl.h"
6
7  #define test 1
8  #define UNKNOWN -1
9  #define CORRECT 0
10 #define FAULTY 1
11
12 typedef struct {
13    int id;
14    int state;
15    int State[10];
16 } TipoProcesso;
17
18 TipoProcesso *processo;
19
20 int main(int argc, char *argv[]) {
21     static int N, token, event, r, i, next;
22     static char fa_name[5];
23
24     if (argc != 2) {
25         puts("Uso correto: tempo <num-processos>");
26         exit(1);
27     }
28
29     N = atoi(argv[1]);
30     smpl(0, "Simula  o com Compartilhamento de Estados");
31
32     processo = (TipoProcesso *) malloc(N * sizeof(TipoProcesso));
33     for (i = 0; i < N; i++) {
34         sprintf(fa_name, "%d", i);
35         processo[i].id = facility(fa_name, 1);
36         processo[i].state = CORRECT;
37         for (int j = 0; j < N; j++) {
```

```
38              processo[i].State[j] = (j == i) ? CORRECT : UNKNOWN;
39          }
40      }
41
42      for (i = 0; i < N; i++) {
43          schedule(test, 30.0, i);
44      }
45
46      while (time() < 100.0) {
47          cause(&event, &token);
48          switch (event) {
49              case test:
50                  next = (token + 1) % N;
51                  r = status(processo[next].id);
52                  processo[token].State[next] = (r == 0) ? CORRECT : FAULTY;
53
54                  if (r == 0) {
55                      for (int j = 0; j < N; j++) {
56                          if (j != token) {
57                          processo[token].State[j] = processo[next].State[j];
58                          }
59                      }
60                      printf("O processo %d compartilhou estados
61                      com o processo %d no tempo %g\n", token, next, time());
62                  }
63
64                  printf("Vetor de Estados do processo %d: ", token);
65                  for (int j = 0; j < N; j++) {
66                      printf("%d ", processo[token].State[j]);
67                  }
68                  printf("\n");
69
70                  schedule(test, 30.0, token);
71                  break;
72          }
73      }
74
75      free(processo);
76      return 0;
77  }
```

```
saulo@Saulo-PC-note:~/Documentos/6oSemestre/CD$ make clean && make tempo && ./tempo 4
rm -f *.o tempo relat saida
cc    -c -o Tempo4.o Tempo4.c
cc    -c  -g -Wno-all smpl.c
cc    -c -g -Wno-all rand.c
cc     -o tempo -Bstatic Tempo4.o smpl.o rand.o -lm
O processo 0 compartilhou estados com o processo 1 no tempo 30
Vetor de Estados do processo 0: 0 0 -1 -1
O processo 1 compartilhou estados com o processo 2 no tempo 30
Vetor de Estados do processo 1: -1 0 0 -1
O processo 2 compartilhou estados com o processo 3 no tempo 30
Vetor de Estados do processo 2: -1 -1 0 0
O processo 3 compartilhou estados com o processo 0 no tempo 30
Vetor de Estados do processo 3: 0 0 -1 0
O processo 0 compartilhou estados com o processo 1 no tempo 60
Vetor de Estados do processo 0: 0 0 0 -1
O processo 1 compartilhou estados com o processo 2 no tempo 60
Vetor de Estados do processo 1: -1 0 0 0
O processo 2 compartilhou estados com o processo 3 no tempo 60
Vetor de Estados do processo 2: 0 0 0 0
O processo 3 compartilhou estados com o processo 0 no tempo 60
Vetor de Estados do processo 3: 0 0 0 0
O processo 0 compartilhou estados com o processo 1 no tempo 90
Vetor de Estados do processo 0: 0 0 0 0
O processo 1 compartilhou estados com o processo 2 no tempo 90
Vetor de Estados do processo 1: 0 0 0 0
O processo 2 compartilhou estados com o processo 3 no tempo 90
Vetor de Estados do processo 2: 0 0 0 0
O processo 3 compartilhou estados com o processo 0 no tempo 90
Vetor de Estados do processo 3: 0 0 0 0
O processo 0 compartilhou estados com o processo 1 no tempo 120
Vetor de Estados do processo 0: 0 0 0 0
```

Figura 5: Print do terminal com a execução do algoritmo da subtarefa 7.4

O makefile utilizado:

```
all: tempo

tempo: Tempo.o smpl.o rand.o
	$(LINK.c) -o $@ -Bstatic Tempo.o smpl.o rand.o -lm

smpl.o: smpl.c smpl.h
	$(COMPILE.c)  -g -Wno-all smpl.c

Tempo.o: Tempo.c smpl.h
	$(COMPILE.c) -g  -Wno-all Tempo.c

rand.o: rand.c
	$(COMPILE.c) -g -Wno-all rand.c

clean:
	$(RM) *.o tempo relat saida
```