RELATÓRIO IMPLEMENTAÇÃO (ESCALONAMENTO DE CPU)

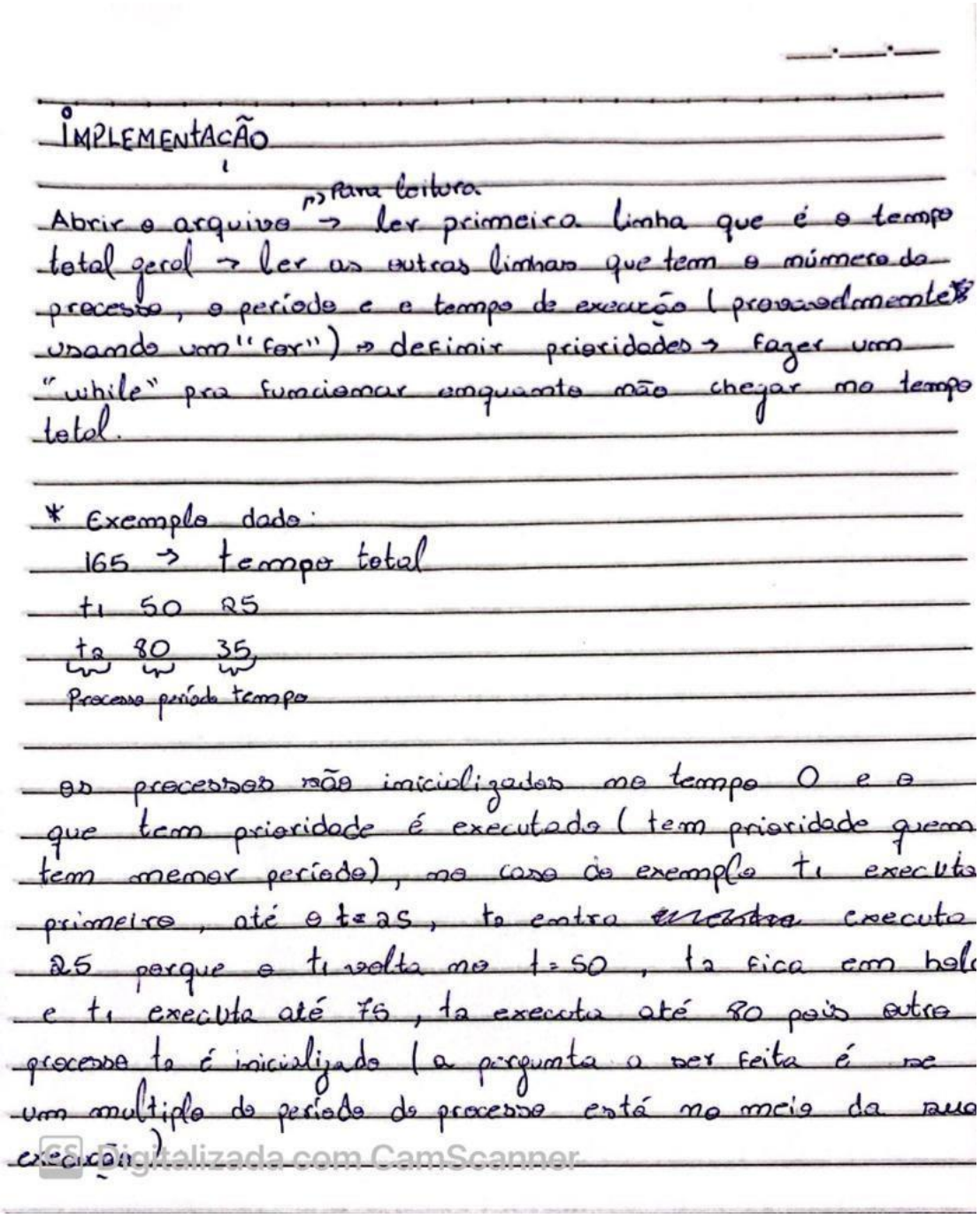
ALUNO: ARTHUR LINS DA GAMA

PERÍODO: 3B

DATA: 03 DE NOVEMBRO DE 2022

INÍCIO:

Antes de começar a implementação tentei escrever à mão em um caderno como seria o passo a passo dela.



IMPLEMENTAÇÃO

Abrir o arquivo (p/ para leitura) → ler primeira linha que é o tempo total geral → ler as outras linhas que tem o número do processo, o período e o tempo de execução (provavelmente usando um "for") → definir prioridades → fazer um "while" pra funcionar enquanto não chegar no tempo total.

* Exemplo dado:

165 → tempo total

t1 50 25

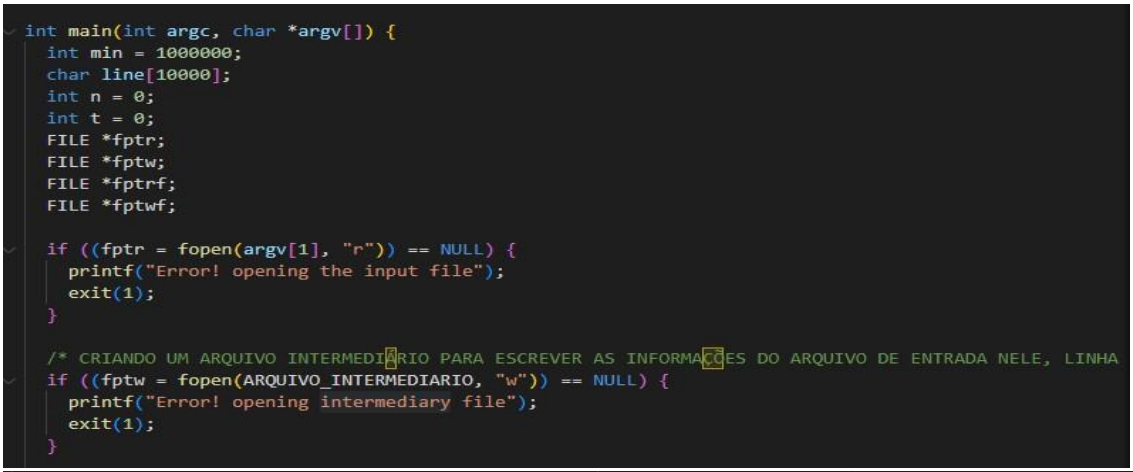
t2 80 35

Processo período tempo

os processos são inicializados no tempo 0 e o que tem prioridade é executado (tem prioridade quem tem menor período), no caso do exemplo t1 executa primeiro, até o t=25, t2 entra executa 25 porque o t1 volta no t=50, t2 fica em hold e t1 executa até 75, t2 executa até 80 pois outro processo t2 é inicializado (a pergunta a ser feita é se um multiplo do período do processo está no meio da sua execução)

LEITURA DO ARQUIVO:

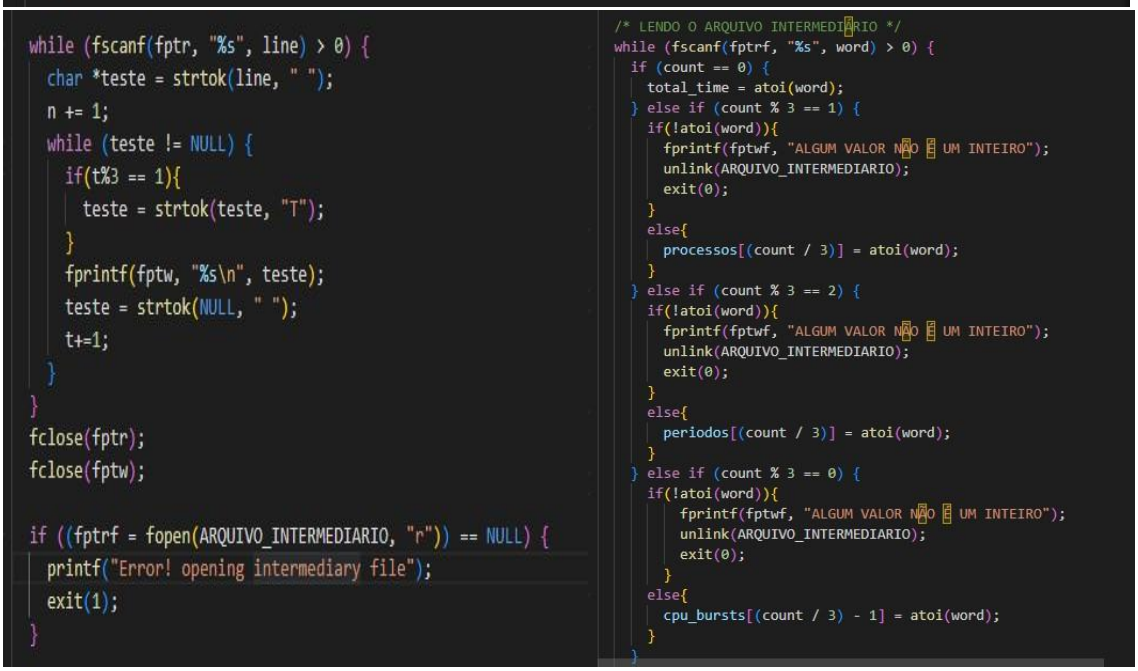
Para ler o arquivo passado pelo usuário, utilizei o padrão dos argumentos, com o número de argumentos sendo representado como o inteiro argc, e os argumentos passados as strings argv[]. Como estava tendo dificuldade em fazer a leitura de 3 elementos na mesma linha, decidi criar um arquivo intermediário que chamei de “correct.txt”, e nele escrevi todos os elementos do arquivo passado linha por linha, sendo assim se no primeiro arquivo havia uma linha com 1 2 3, o arquivo intermediário teria uma linha com o número 1, uma com o número 2 e uma com o número 3. Após isso, fiz a leitura desse segundo arquivo, li as linhas como char, porém utilizei da função atoi() para transformálas em inteiros. Sendo a primeira linha o tempo total da execução, e utilizei um count que seria a contagem da quantidade de linhas até o fim do arquivo e fiz que quando o resto da divisão do count por 3 fosse 1, o valor dessa linha seria o processo, quando fosse 2 seria o período do processo e quando fosse 3 o burst. Ao final da leitura apaguei o arquivo intermediário para não ocupar nenhum espaço.



```c
int main(int argc, char *argv[]) {
  int min = 1000000;
  char line[10000];
  int n = 0;
  int t = 0;
  FILE *fptr;
  FILE *fptw;
  FILE *fptrf;
  FILE *fptwf;

  if ((fptr = fopen(argv[1], "r")) == NULL) {
    printf("Error! opening the input file");
    exit(1);
  }

  /* CRIANDO UM ARQUIVO INTERMEDIÁRIO PARA ESCREVER AS INFORMAÇÕES DO ARQUIVO DE ENTRADA NELE, LINHA
  if ((fptw = fopen(ARQUIVO_INTERMEDIARIO, "w")) == NULL) {
    printf("Error! opening intermediary file");
    exit(1);
  }
```



```c
while (fscanf(fptr, "%s", line) > 0) {
  char *teste = strtok(line, " ");
  n += 1;
  while (teste != NULL) {
    if(t%3 == 1){
      teste = strtok(teste, "T");
    }
    fprintf(fptw, "%s\n", teste);
    teste = strtok(NULL, " ");
    t+=1;
  }
}
fclose(fptr);
fclose(fptw);

if ((fptrf = fopen(ARQUIVO_INTERMEDIARIO, "r")) == NULL) {
  printf("Error! opening intermediary file");
  exit(1);
}
```

```c
/* LENDO O ARQUIVO INTERMEDIÁRIO */
while (fscanf(fptrf, "%s", word) > 0) {
  if (count == 0) {
    total_time = atoi(word);
  } else if (count % 3 == 1) {
    if(!atoi(word)){
      fprintf(fptwf, "ALGUM VALOR NÃO É UM INTEIRO");
      unlink(ARQUIVO_INTERMEDIARIO);
      exit(0);
    }
    else{
      processos[(count / 3)] = atoi(word);
    }
  } else if (count % 3 == 2) {
    if(!atoi(word)){
      fprintf(fptwf, "ALGUM VALOR NÃO É UM INTEIRO");
      unlink(ARQUIVO_INTERMEDIARIO);
      exit(0);
    }
    else{
      periodos[(count / 3)] = atoi(word);
    }
  } else if (count % 3 == 0) {
    if(!atoi(word)){
      fprintf(fptwf, "ALGUM VALOR NÃO É UM INTEIRO");
      unlink(ARQUIVO_INTERMEDIARIO);
      exit(0);
    }
    else{
      cpu_bursts[(count / 3) - 1] = atoi(word);
    }
  }
```

TRATAMENTO DE ERROS NA LEITURA DOS ARQUIVOS:

Caso os arquivos não pudessem ser abertos imprimi a mensagem “Error! Opening the [arquivo] file”. Exemplo:

```
if ((fptr = fopen(argv[1], "r")) == NULL) {
  printf("Error! opening the input file");
  exit(1);
}
```

Caso o usuário passe mais de 1 argumento eu imprimi a mensagem “QUANTIDADE ERRADA DE ARGUMENTOS, TENTE APENAS COM UM ARQUIVO! ”. Exemplo:

```
if(argc > 2){
    fprintf(fptwf, "QUANTIDADE ERRADA DE ARGUMENTOS, TENTE APENAS COM UM ARQUIVO!");
    unlink(ARQUIVO_INTERMEDIARIO);
    exit(0);
}
```

Caso algum valor dado no arquivo não pudesse ser convertido em inteiro imprimi a mensagem “ALGUM VALOR NÃO PODE SER CONVERTIDO EM INTEIRO”. Exemplo:

```
if(!atoi(word)){
  fprintf(fptwf, "ALGUM VALOR NÃO PODE SER CONVERTIDO EM INTEIRO");
  unlink(ARQUIVO_INTERMEDIARIO);
  exit(0);
}
```

Caso o arquivo não possua o número correto de valores que deveriam ser passados eu imprimi a mensagem “ALGUMA INFORMAÇÃO AUSÊNTE NO ARQUIVO DE LEITURA, POR FAVOR TENTAR COM OUTRO! ”. Exemplo:

```
}
if((count - 1)%3 != 0){
  fprintf(fptwf, "ALGUMA INFORMAÇÃO AUSÊNTE NO ARQUIVO DE LEITURA, POR FAVOR TENTAR COM OUTRO!");
  unlink(ARQUIVO_INTERMEDIARIO);
  exit(0);
}
```

ALGORITMO:

De começo inicializei todas as variáveis como elas devem ser:

```
for (int j = 0; j < count / 3; j++) {
  init[j] = 1;
  holds[j] = 0;
  hold[j] = 0;
  complete[j] = 0;
  lost[j] = 0;
  units_comp[j] = 0;
}
```

```
for(int i = 0; i < count / 3; i++){
  sum += init[i];
}
```

```
else if(sum == 0){
  idle += 1;
  total_time_cp += 1;
  for(int i = 0; i < count / 3; i++){
    if(total_time_cp%periodos[i] == 0){
      init[i] = 1;
    }
  }
}
```

Em seguida comecei a implementação do algoritmo, decidi começar checando qual o menor período de todos os processos, pois os menores períodos têm a prioridade, com isso a execução sempre será do processo inicializado que tenha menor período. E caso nenhum processo fosse inicializado criei uma variável chama sum para checar isso, pois todos os processos inicializados foram atribuídos em um array com o valor de 1, caso a soma desse array inteiro fosse 0 seria porque nenhum processo está para ser executado, sendo o caso de contar quantas unidades de idle tiveram nesse momento.

```
/* DEFINIR PRIORIDADES */
int sum = 0;
int min = 1000000;
for(int m = 0; m < count / 3; m++){
  if(init[m] == 1){
    if(periodos[m] < min){
      min = periodos[m];
      processo = processos[m];
      periodo = periodos[m];
      burst = cpu_bursts[m];
      index = m;
    }
  }
}
```

Então fiz os casos requisitados, quando um processo atingir o seu tempo de burst ele é finalizado imprimindo quantas unidades executou e um “-F” que representa que foi finalizado. Caso ele ainda esteja sendo executado e chegue o tempo de seu período ele é perdido, o que seria o lost deadline, imprimindo quantas unidades executou e um “-L”, porém se houver um lost deadline em um processo que não está executando, isso não é impresso, mas é contabilizado. E quando um processo estiver em execução e chegar um processo que tenha maior prioridade, ele fica em estado de hold, imprimindo quantas unidades executou e um “H”, e guardando quantas unidades executou em um array para quando for executar de novo já iniciar do ponto que parou. Por último, caso chegue ao tempo total e alguns processos ainda estejam inicializados, esses processos serão mortos, e caso o que esteja em execução seja morto será impresso a quantidade de unidades que ele executou e um “-K” se referindo a killed.

```c
/* CASO NENHUM PROCESSO ESTEJA SENDO EXECUTADO */
if(idle > 0){
  fprintf(fptwf, "idle for %d units\n", idle);
  idle = 0;
  }

  total_time_cp += 1;
  units_comp[index] += 1;

  if(units_comp[index]%burst == 0){
    /* CASO O PROCESSO SEJA FINALIZADO */
     fprintf(fptwf, "[T%d] for %d units - F\n", processo, units_comp[index]-holds[index]);
     complete[index] += 1;
     init[index] = 0;
     holds[index] = 0;
     units_comp[index] = 0;
    }

  else{
    /* CONTAR OS LOST DEADLINES DE PROCESSOS QUE ESTEJAM EXECUTANDO */
    if(total_time_cp%periodos[index] == 0 && units_comp[index] != cpu_bursts[index] && units_comp[index] > 0){
      fprintf(fptwf, "[T%d] for %d units - L\n", processo, units_comp[index] - holds[index]);
           units_comp[index] = 0;
           holds[index] = 0;
           lost[index] += 1;
           init[index] = 1;
       }
    for(int i = 0; i < count / 3; i++){

      /* CONTAR OS LOST DEADLINES MESMO QUE O PROCESSO NAO ESTEJA EXECUTANDO */
      if(init[i] == 1 && total_time_cp%periodos[i] == 0 && units_comp[i] != cpu_bursts[i] && i != index && hold[i] == 1){
           units_comp[i] = 0;
           holds[i] = 0;
           hold[i] = 0;
           lost[i] += 1;
           init[i] = 1;
       }

      else if(total_time_cp%periodos[i] == 0){
        /* CASO O PROCESSO FIQUE EM HOLD */
        if(init[i] == 0 && i != index && periodos[i] < min && units_comp[index] > 0){
          fprintf(fptwf, "[T%d] for %d units - H\n", processo, units_comp[index] - holds[index]);
              holds[index] = units_comp[index];
              hold[index] = 1;
              units_comp[i] = 0;
              init[i] = 1;
          }
```

Tudo isso sendo executado enquanto o tempo contado não chegar no tempo total.

IMPRIMINDO OS RESULTADOS:

Ao final de tudo imprimi os resultados do escalonamento, dos lost deadlines, dos processos completados e dos mortos ao final da execução, respectivamente, como exigido:

```c
/* IMPRIMIR RESULTADOS */
fprintf(fptwf, "\nLOST DEADLINES\n");
for(int n = 0; n < count / 3; n++){
    fprintf(fptwf, "[T%d] %d\n", processos[n], lost[n]);
  }
fprintf(fptwf, "\nCOMPLETE EXECUTION\n");
for(int n = 0; n < count / 3; n++){
    fprintf(fptwf, "[T%d] %d\n", processos[n], complete[n]);
  }
fprintf(fptwf, "\nKILLED\n");
for(int n = 0; n < count / 3; n++){
  if(init[n] == 1){
    fprintf(fptwf, "[T%d] 1\n", processos[n]);
  }
  else{
    fprintf(fptwf, "[T%d] 0\n", processos[n]);
  }
}
```

DIFICULDADES:

Felizmente consegui finalizar com rapidez a implementação, porém tive muita dificulade com a leitura do arquivo sendo mais de um elemento em uma única linha, por isso a criação de um arquivo intermediário.

Outro problema foi quando após “finalizar” o código, fui testar e percebi que além de não estar contando lost deadlines de processos que não estavam executando, o código estava rodando 2 processos ao mesmo tempo caso o período dos 2 fossem múltiplos, por exemplo T1 tem período de 40 e T2 de 80, quando o tempo chegasse em 80 os 2 executavam, somando suas unidades no tempo ao mesmo tempo, isso mesmo o período de T1 sendo menor, ou seja, ele supostamente teria prioridade, então se o tempo de burst deles fossem 25 e 35 respectivamente, e o tempo total 165, ao final se fossem contar as unidades executadas a soma daria 185, o que passava do tempo total. Porém consegui resolver utilizando um array com as unidades executadas em cada processo, e o problema do lost deadlines, resolvi fazendo uma condição dentro de um for em que o “i” do for fosse diferente do index do processo que estava executando.

RESULTADO:

```
EXECUTION BY RATE
[T1] for 25 units - F
[T2] for 25 units - H
[T1] for 25 units - F
[T2] for 5 units - L
[T2] for 20 units - H
[T1] for 25 units - F
[T2] for 15 units - F
idle for 10 units
[T1] for 15 units - K

LOST DEADLINES
[T1] 0
[T2] 1

COMPLETE EXECUTION
[T1] 3
[T2] 1

KILLED
[T1] 1
[T2] 1
```