Descrição do trabalho 1:

Criar um programa que possua pelo menos duas threads, onde uma faz a “Recepção” de processos “clientes” (criando-os), colocando-os em uma fila de atendimento, e a outra thread realiza o trabalho do “atendente”, retirando-os da fila. O processo “Cliente” será dado pronto, enquanto o processo “Atendimento”, além das threads já descritas, pode ter mais threads para fazer o devido controle da fila (ou pode ser usada uma das duas threads originais.

O processo “Atendimento” deve receber dois valores de entrada: N, que é o número de clientes a serem gerados pela thread “Recepção” (sendo 0 para infinitos), e X, que é o tempo máximo, em ms, que um cliente tem “paciência” para esperar. Os clientes podem ser de dois tipos: prioridade alta, que tem a paciência de X/2 ms, e prioridade baixa, que tem a paciência de X ms. Um cliente se dá por “satisfeito” quando seu atendimento é completado em tempo inferior ou igual à sua paciência. O processo “Atendimento”, então, termina quando atende a todos os clientes e deve retornar a taxa de satisfação (quantidade de clientes satisfeitos/pelo total de clientes) e o tempo total de execução. Para o caso de N igual a 0, o processo atendimento deve parar quando a tecla “s” for clicada, retornando a taxa de satisfação levando em consideração apenas os clientes que foram atendidos (ignora-se os que ficaram na fila para esse cálculo).

A atribuição de prioridades aos clientes é feita pela thread “Recepção” ao criar um processo “Cliente”. Essa atribuição deve ser aleatória com probabilidade de 50% de atribuição de cada prioridade.

Quando um cliente for atendido, seu PID deve ser escrito em um arquivo a ser lido por um processo “Analista” que imprimirá os números na tela. O processo analista deve ser implementado de tal forma que, no máximo, os 10 primeiros valores do arquivo sejam impressos toda vez que “acordar” (e apagados do arquivo). Ele deve ser criado “dormindo” e deve ser “acordado” pela thread “Atendente” sempre que você achar necessário imprimir. IMPORTANTE: no caso de número infinito de clientes, que o usuário pode fechar o processo de atendimento a qualquer momento apertando “s”, o arquivo deve ser esvaziado antes, isto é, o “Analista” deve ser acordado tantas vezes quantas forem necessárias para imprimir todos os PID de clientes que já haviam sido atendidos.

O trabalho deve ser necessariamente feito em Linux usando a linguagem C. Não é permitido o uso de bibliotecas que ajudem no paralelismo ou na comunicação entre processos. As threads devem ser criadas usando pthreads. Para mais detalhes, observe o diagrama em anexo.

Avaliação:

1. Pode ser feito individual ou em dupla;

2. A nota será dada em duas partes: uma pontuação em comum para a equipe e uma pontuação a ser dividida pela própria equipe entre seus integrantes de acordo com a “quantidade de trabalho” que considerarem realizada por cada um;

3. Um integrante da equipe pode ficar com mais de 10,0 e os pontos que passarem serão guardados para o próximo trabalho;

4. As notas serão “competitivas”. O trabalho com o melhor desempenho da turma (medido pela taxa de satisfação e pelo tempo de execução) tem 10,0 e aquele com o pior desempenho tem 7,0. Os desempenhos intermediários terão suas notas definidas por interpolação linear. Trabalhos que não

funcionarem ficarão com nota abaixo de 7,0 e serão avaliados caso a caso através do “aproveitamento” de código;

5. No caso de empate, ou seja, dois ou mais trabalhos com desempenhos exatamente iguais, a nota será 7,0 mais um “fator de identidade”, calculado da seguinte forma:

5.1. A cada conjunto com K trabalhos de desempenhos idênticos será atribuída uma mesma nota parcial Np calculada de acordo com o passo 4;

5.2. O “fator de identidade” I, será: **I = (Np – 7,0)/K**

5.3. A nota final N desses trabalhos será: **N = 7,0 + I**

6. Os trabalhos serão apresentados nos dias 19 e 20 de Dezembro. No dia 19 todas as equipes devem estar com seus trabalhos prontos, pois a chamada será aleatória. Mais ou menos metade da turma será chamada a apresentar no dia 19 e o restante no dia 20. Caso haja alguma impossibilidade de presença, ela deve ser comunicada e justificada. Se alguma equipe quiser se candidatar para apresentar primeiro, pode fazê-lo. Não é necessária a presença no dia 20 para quem tiver apresentado já no dia 19;

7. Para a apresentação, além de mostrar o trabalho rodando, a equipe deve preparar slides com a explicação das estratégias aplicadas no trabalho. Esses slides devem ser submetidos pelo SIGAA até as 13h do dia 19 de dezembro, independentemente do dia da apresentação da equipe.

Processo 1 - Atendimento[8]

Thread 1 - Atendente

- Lê na fila próximo cliente
- Acorda o cliente
- Espera semáforo "\sem_atend" abrir
- Espera semáforo "\sem_block" abrir
- Fecha semáforo "\sem_block"
- Escreve o número do cliente (PID) no arquivo Lista de Números Gerados (LNG)
- Abre semáforo "\sem_block"
- Calcula satisfação[6] do cliente
- Tira cliente da fila
- Acorda Analista[1]

Fila[2] de clientes

Thread 2 - Recepção

- Cria semáforos "\sem_atend" e "\sem_block"
- Cria N ou infinitos processos clientes[9]
- Cada processo é criado com uma prioridade[3] entre duas possíveis
- Prioridade aleatória: 50% para cada

Estrutura do cliente na fila

- PID
- Hora de chegada[4]
- Prioridade
- Tempo para atendimento

Arquivo LNG

- Bloqueável pelo semáforo "\sem_block"

Processo 2 - Analista

- Gera arquivo com seu PID e dorme
- Quando acordado:
- Bloqueia LNG
- Lê LNG e imprime seus 10 primeiros valores
- Apaga os 10 valores que imprimiu
- Desbloqueia LNG
- Dorme novamente

Processo 3 - Cliente[5]

- Gera e escreve seu tempo necessário para atendimento no arquivo Demanda
- Dorme
- Quando acordado: fecha semáforo de atendimento "\sem_atend" e pausa pela quantidade de tempo de atendimento calculado
- Abre semáforo "\sem_atend" de atendimento
- Finaliza

Arquivo Demanda

- Escrito pelo cliente na geração
- Lido pelo Atendimento para colocar na fila
- Após lido, pode ser apagado[7]

1 - A estratégia para acordar o analista, se é a cada atendimento, se é periódico por tempo, por quantidade de atendimentos ou qualquer outra estratégia, é escolha sua;
2 - A estrutura de dados da "fila" é uma escolha de projeto sua. Não precisa ser literalmente uma fila e pode ser mais de uma; A inserção de clientes na fila pode ser feita pela thread 2 ou uma nova thread pode ser criada só para tratamento da fila;
3 - Prioridade baixa significa cliente que tem X ms de "paciência". Prioridade alta significa cliente que tem X/2 ms de "paciência". O valor de X é dado de entrada para o Processo "Atendimento";
4 - A hora de chegada é o tempo em ms passados entre o início do programa e a criação do cliente;
5 - Será dado. Todas as equipes utilizarão o mesmo processo cliente;
6 - A satisfação do cliente é "Satisfeito" se (tempo atual - hora de chegada) <= paciência. Caso contrário é "Insatisfeito";
7 - O arquivo Demanda pode ser único e serve para comunicação entre os processos "Cliente" e "Atendimento" apenas no momento da inserção na fila. Como os clientes são colocados na fila um por vez, esse arquivo é compartilhado entre todos os clientes;
8 - O processo Atendimento recebe de entrada o valor N de número de clientes a serem criados e o valor X de paciência de prioridade baixa. Ele dá de saída (imprime na tela) a taxa de satisfação e o tempo total de execução. Ele termina quando todos os N clientes tiverem sido atendidos.
9 - Quando o N passado for 0, a Recepcão deve ficar em loop infinito criando clientes. Ela deve pausar a criacão se a fila ficar com 100 clientes em espera, voltando a criar mais toda vez que um espaco for liberado na fila.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <signal.h>
#include <semaphore.h>
#include <fcntl.h>

int main()
{
    int tempo;
    srand( (unsigned)time(NULL) );
    int x = rand()%10;

    if(x == 0)
        tempo = 15;
    else if(x > 0 && x <= 3)
        tempo = 5;
    else
        tempo = 1;

    FILE* demanda = fopen("demanda.txt", "w+");
    fprintf(demanda, "%d", tempo);
    fclose(demanda);

    raise(SIGSTOP);

    sem_t *sem = sem_open("/sem_atend", O_RDWR);

    if(sem != SEM_FAILED) sem_wait(sem);

    usleep(tempo);

    if(sem != SEM_FAILED) sem_post(sem);

    return 0;

}
```