*ALUNO: JOÃO GABRIEL CARNEIRO MEDEIROS.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
CEARÁ

Departamento de Telemática
Avaliação de Software e Sistema de Tempo-Real
Prof. Paulo Régis C. de Araújo

Nome:

1. Elabore, em linguagem C, duas funções para comunicação entre threads: uma função de envio sincronizado e temporizado (int send_sync_timeout(int *buf, int c, float tempo)) e uma função de recepção temporizada ((int receive_timeout(int *buf, int c, float tempo)). Utilize o conceito de busy wait loop e flag para a sincronização. (6 escores)

2. Elabore o escopo de recebimento de votos do processo driver, de um programa com 3 versões, utilizando o conceito de espera seletiva. Obs: não precisa apresentar o envio do status de comparação realizado pelo processo driver. (6 escores)

3. Quatro threads (Ta(), Tb(), Tc() e Td()) devem enviar mensagens, por canal de comunicação sincronizado, para um processo destino chamado de driver. A thread Ta(), antes de enviar sua mensagem, aguarda em uma função de atraso 12 segundos. A thread Tb(), antes de enviar sua mensagem, aguarda em uma função de atraso 6 segundos. A thread Tc(), antes de enviar sua mensagem, aguarda em uma função de atraso 20 segundos. E, por fim, a thread Td(), antes de enviar sua mensagem, aguarda em uma função de atraso 1 segundo. Se a thread driver realizar o recebimento das mensagens das threads Ta(), Tb(), Tc() e Td() sequencialmente através de 4 funções de recepção (ex. receive(&x, 0); // para Ta, receive(&x, 1); // para Tb, receive(&x, 2); // para Tc, receive(&x, 3) para Td, depois de quantos segundos a thread driver receberá o valor enviado pela thread Td()? Explique sua resposta. (4 escores)

*Aluno: JOÃO GABRIEL CARNEIRO MEDEIROS

```
int send_sync_timed(int *buffer, int c, float tempo) {
    timed_t inicio, fim;
    canal[c] = *buffer;
    inicio = time(null);
    do {
        fim = time(null);
    } while((canal[c] != -1) && (difftime(fim, inicio) <= tempo))
    if(canal[c] == -1){
        return 0;
    } else {
        return 1;
    }
}

int receive_timed(int* buffer, int c, float tempo) {
    time_t inicio, fim;
    inicio = time(null);
    do {
        fim = time(null);
    } while((canal[c] == -1) && (difftime(fim, inicio) <= tempo))
    if(canal[c] != -1) {
        *buffer = canal[c];
        canal[c] = -1;
        return 0;
    } else {
        return 1;
    }
}
```

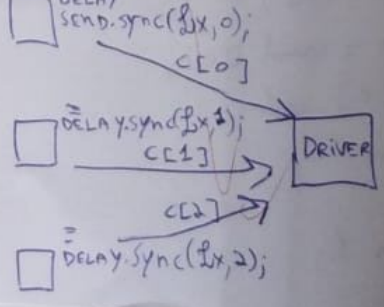
02) PODEMOS FAZER A SEGUINTE ANÁLISE:

i)
```
INT CANAL[3] = {-1,-1,-1};

VOID DRIVER(){
    INT VETOR[3], INT I, INT INDICE;
    FOR (I=0; I<3; ++I){
        INDICE = ALT_WAIT(3, CANAL);
        RECEIVE(&VETOR[I], INDICE);
    }
}
```

ii) → *OBS:
```
INT ALT_WAIT(INT Q, INT VET[]){
    INT X = 0
    WHILE(1){
        FOR(X=0; X<Q; ++X){
            IF(VET[X] != -1){
                return X;
            }
        }
    }
    return 0;
}
```

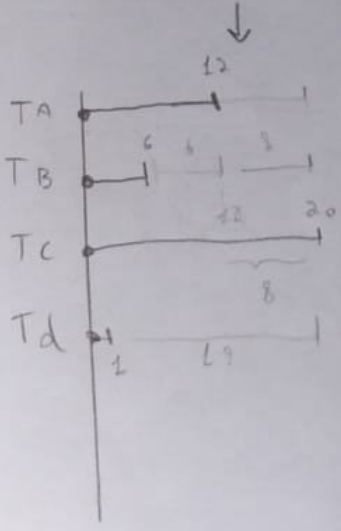
iii) → ONDE:



5) i) SABENDO QUE, NA THREAD "DRIVER" TEMOS:
*(SEQUENCIALMENTE):

ATRASO

Receive(&x, 0); // PARA TA (AGUARDA 12s)
Receive(&x, 1); // PARA Tb (AGUARDA 6s)
Receive(&x, 2); // PARA Tc (AGUARDA 20s)
Receive(&x, 3); // PARA Td (AGUARDA 1s)



ii) DEPOIS DE 20 SEGUNDOS, POIS A Td() APÓS SER CHAMADA POR ÚLTIMO JÁ ESPERA SEU 1 SEGUNDO E SÓ É CHAMADA DEPOIS DE Tc();

↳ NOTE QUE, MESMO JÁ SE PASSANDO TA E Tb AINDA FORAM 12s! MAIS O RESTO DE Tc() FICAM OS 20s ATÉ Td() ENVIAR SUA MENSAGEM!