Sistemas Operacionais
Atividade 1

Guilherme Sales Fernandes - 499756

Estrutura Geral do Sistema

client.c

service.c

analyst.c

Estrutura Geral do Sistema (service)

```c
// Estrutura para representar um cliente
typedef struct {
    int pid;
    long chegada;
    int prioridade;
    long tempo_para_atendimento;
} Cliente;

// Fila Circular
Cliente fila[TAM_FILA];
int inicio = 0, fim = 0, tamanho = 0;
```

Fila circular: reutilizar os mesmos espaços na memória , evitando a necessidade de realocação dinâmica ou expansão da estrutura (apenas atualizar os índices). Dessa forma, quando um cliente é atendido e removido da fila, o espaço ocupado por ele fica imediatamente disponível.

Estrutura Geral do Sistema (service)

```c
// Funções Auxiliares
void insere_fila(Cliente c) {
    fila[fim] = c;
    fim = (fim + 1) % TAM_FILA;
    tamanho++;
}

Cliente remove_fila() {
    Cliente c = fila[inicio];
    inicio = (inicio + 1) % TAM_FILA;
    tamanho--;
    return c;
}

long timestamp_ms() {
    struct timespec ts;
    clock_gettime(CLOCK_MONOTONIC, &ts);
    return ts.tv_sec * 1000 + ts.tv_nsec / 1000000;
}
```

- `insere_fila`: Adiciona um cliente no final da fila.
- `remove_fila`: Remove o cliente mais antigo.
- `timestamp_ms`: Calcula o timestamp atual em milissegundos para controlar tempos de chegada e atendimento.

Estrutura Geral do Sistema (service)

```c
// Thread 2 - Recepção
void *recepcao(void *args) {
    int N = *((int *)args);
    int count = 0;

    printf("SERVICE: Recepção: Criando semáforo /sem_atend\n");
    sem_atend = sem_open("/sem_atend", O_CREAT, 0644, 0);
    if (sem_atend == SEM_FAILED) {
        perror("Erro ao criar semáforo /sem_atend");
        exit(1);
    }

    printf("SERVICE: Recepção: Criando semáforo /sem_block\n");
    sem_block = sem_open("/sem_block", O_CREAT, 0644, 1);
    if (sem_block == SEM_FAILED) {
        perror("Erro ao criar semáforo /sem_block");
        exit(1);
    }
    while (N == 0 || (N > 0 && count < N)) {
        printf("SERVICE: Recepção: Aguardando semáforo /sem_espaco\n");
        sem_wait(&sem_espaco);
        printf("SERVICE: Recepção:  Semáforo /sem_espaco informa que tem espaço\n");
        inspecionar_semaforo(sem_atend, "                    -                                    /sem_atend");
        pid_t pid = fork();
        if (pid > 0) {
            printf("SERVICE: Recepção: Cliente %d criado com PID: %d\n", count+1, pid);
        }
        if (pid == 0) {
            execl("./client", "./client", NULL);
            perror("Erro ao executar client");
            exit(1);
        } else if (pid > 0) {
            printf("SERVICE: Recepção: Esperando semáforo /sem_atend\n");
            sem_wait(sem_atend);
            printf("SERVICE: Recepção: Semáforo /sem_atend liberado\n");

            int tempo_demanda = ler_demanda();
            printf("Recepção: Tempo lido do arquivo demanda.txt: %d\n", tempo_demanda);

            Cliente cliente;
            cliente.pid = pid;
            cliente.chegada = timestamp_ms();
            cliente.prioridade = (rand() % 2 == 0) ? 1 : 0;
```

```c
            cliente.tempo_para_atendimento = (cliente.prioridade == 1) ? tempo_demanda/2 : tempo_demanda;

            sem_wait(&sem_fila);
            insere_fila(cliente);
            sem_post(&sem_fila);

            sem_post(&sem_clientes);
            if (N > 0){
                count++;
            }
            else {
                count = 0;
            }
        }
    }
    pthread_exit(NULL);
}
```

Semáforos:

- `sem_espaco`: Verifica se há espaço na fila.
- `sem_clientes`: Sinaliza que um cliente foi adicionado à fila.

Ações:

- Cria clientes com `fork` e executa o código cliente com `execl`.
- Gera prioridades e tempos de paciência aleatórios.
- Insere o cliente na fila protegida por `sem_fila`.

Estrutura Geral do Sistema (service)

```c
// Thread 1 - Atendente
void *atendimento(void *args) {
    pid_t pid_analista = *((pid_t *)args);

    while (1) {
        printf("SERVICE: Atendimento: Aguardando cliente...\n");
        sem_wait(&sem_clientes); // Espera até que haja clientes na fila

        printf("SERVICE: Atendimento: Aguardando liberação da fila\n");
        sem_wait(&sem_fila);

        Cliente cliente = remove_fila(); // Remove o cliente da fila
        printf("SERVICE: Atendimento: Cliente %d removido da fila\n", cliente.pid);
        sem_post(&sem_fila);

        sem_wait(sem_atend); // Sincronização de atendimento
        sem_wait(sem_block); // Bloqueia o acesso ao arquivo

        long tempo_atual = timestamp_ms();
        int satisfeito = (tempo_atual - cliente.chegada) <= cliente.tempo_para_atendimento;

        // Atualiza métricas protegidas pelo semáforo
        sem_wait(&sem_metrics);
        total_clientes++;
        if (satisfeito) {
            clientes_satisfeitos++;
        } else {
            clientes_insatisfeitos++;
        }

        // Calcula a taxa de satisfação e o tempo total de execução
        float taxa_satisfacao = (total_clientes > 0) ?
                                ((float)clientes_satisfeitos / total_clientes) * 100 : 0.0;
        long tempo_total_execucao = timestamp_ms() - inicio_execucao;
        printf("--");
        printf("Tempo atual: %ld\n", timestamp_ms());
        printf("Tempo de chegada: %ld\n", inicio_execucao);
        printf("--");
        // Atualiza o arquivo de métricas
        FILE *metrics_file = fopen("metrics.txt", "w");
        if (metrics_file != NULL) {
            fprintf(metrics_file, "Total de clientes: %d\n", total_clientes);
            fprintf(metrics_file, "Clientes satisfeitos: %d\n", clientes_satisfeitos);
            fprintf(metrics_file, "Clientes insatisfeitos: %d\n", clientes_insatisfeitos);
            fprintf(metrics_file, "Taxa de satisfação: %.2f%%\n", taxa_satisfacao);
            fprintf(metrics_file, "Tempo total de execução: %ld ms\n", tempo_total_execucao);
            fclose(metrics_file);
        }
        sem_post(&sem_metrics);
```

```c

        // Escreve os dados do cliente no arquivo
        FILE *arquivo = fopen(LNG_FILE, "a");
        if (arquivo != NULL) {
            fprintf(arquivo, "[
            fclose(arquivo);
        }

        sem_post(sem_block); // Libera o acesso ao arquivo
        printf("SERVICE: Atendimento do cliente %d concluído.\n", cliente.pid);

        // Acorda o analista imediatamente após concluir o atendimento
        printf("SERVICE: Atendimento: Acordando analista de PID: %d...\n", pid_analista);
        sem_post(sem_analyst_ready); // Libera o analista
        kill(pid_analista, SIGCONT); // Envia o sinal SIGCONT para o analista
        printf("SERVICE: Atendimento: Enviando sinal SIGCONT para analista de PID %d\n", pid_analista);
    }
}
```

- **Semáforos**:
  - `sem_clientes`: Garante que há clientes na fila.
  - `sem_fila`: Protege a remoção de clientes da fila.
  - `sem_block`: Bloqueia o acesso ao arquivo enquanto é atualizado.
  - `sem_metrics`: Protege a atualização das métricas.
- **Ações**:
  - Remove o cliente da fila e verifica se foi atendido dentro do tempo de paciência.
  - Atualiza o arquivo `LNG.txt` com os resultados do atendimento.
  - Acorda o analista após processar um cliente.

Estrutura Geral do Sistema (service)

```c
int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Uso: %s <N>\n", argv[0]);
        return 1;
    }

    int N = atoi(argv[1]);
    srand(time(NULL));

    inicio_execucao = timestamp_ms();

    // Remover semáforos antigos
    sem_unlink("/sem_atend");
    sem_unlink("/sem_block");
    sem_unlink("/sem_analyst_ready");

    // Limpar arquivos
    FILE *file = fopen(LNG_FILE, "w");
    if (file) fclose(file);

    // Inicializar Semáforos
    sem_init(&sem_fila, 0, 1);
    sem_init(&sem_espaco, 0, TAM_FILA);
    sem_init(&sem_clientes, 0, 0);
    sem_init(&sem_metrics, 0, 1);
    sem_analyst_ready = sem_open("/sem_analyst_ready", O_CREAT, 0644, 0);

    // Criar o processo Analista
    pid_t pid_analista = start_analista();

    // Criar threads
    pthread_t th_recepcao, th_atendimento;
    pthread_create(&th_recepcao, NULL, recepcao, &N);
    pthread_create(&th_atendimento, NULL, atendimento, &pid_analista);
```

```c
// Aguardar threads
pthread_join(th_recepcao, NULL);
pthread_cancel(th_atendimento);

// Destruir Semáforos
sem_destroy(&sem_fila);
sem_destroy(&sem_espaco);
sem_destroy(&sem_clientes);
sem_close(sem_atend);
sem_close(sem_block);
sem_unlink("/sem_block");
sem_close(sem_analyst_ready);
sem_unlink("/sem_analyst_ready");
sem_destroy(&sem_metrics);
```

Estrutura Geral do Sistema (analyst)

```c
void process_lng_file() {
    FILE *file = fopen(LNG_FILE, "r+");
    if (file == NULL) {
        perror("Erro ao abrir LNG");
        return;
    }

    char buffer[1024];
    char restante[1024];
    int count = 0;

    restante[0] = '\0';

    printf("Valores lidos do arquivo LNG:\n");
    while (fgets(buffer, sizeof(buffer), file) != NULL) {
        if (count < 10) {
            printf("%s", buffer);
            count++;
        } else {
            strcat(restante, buffer);
        }
    }

    freopen(LNG_FILE, "w", file);
    fprintf(file, "%s", restante);
    fclose(file);
}
```

- **Ações**:
  - Lê e imprime até 10 registros por vez.
  - Remove os registros processados do arquivo.
  - Na prática, ele lê 1 registro e remove esse registro, já que ele é chamado sempre que um cliente é atendido.

Estrutura Geral do Sistema (analyst)

```c
int main() {
    sem_analyst_ready = sem_open("/sem_analyst_ready", O_CREAT, 0644, 0);

    signal(SIGCONT, handle_signal);

    printf("ANALYST: Processo Analista iniciado. PID: %d\n", getpid());

    // Cria ou abre o semáforo \sem_block
    sem_t *sem_block = sem_open("/sem_block", O_CREAT, 0644, 1);
    if (sem_block == SEM_FAILED) {
        perror("Erro ao criar o semáforo sem_block");
        exit(1);
    }
    sem_t *sem_clientes = sem_open("/sem_clientes", O_CREAT, 0644, 0);
    if (sem_clientes == SEM_FAILED) {
        perror("Erro ao criar o semáforo sem_clientes");
        exit(1);
    }
    while (1) {
        printf("ANALYST: Analista dormindo...\n");

        raise(SIGSTOP);
        printf("ANALYST: Analista recebeu um sinal. Desbloqueando...\n");

        printf("ANALYST: Antes de bloquear sem_block\n");
        sem_wait(sem_block);
        printf("ANALYST: Semáforo sem_block bloqueado\n");

        process_lng_file();

        sem_post(sem_block);
        printf("ANALYST: Semáforo sem_block liberado\n");


        printf("ANALYST: Analista voltou a dormir...\n");
    }
```

- sem_analyst_ready é usado pelo atendente para sinalizar que o Analista pode acordar e processar os dados.
- sem_block protege o acesso ao arquivo `LNG.txt` contra acessos simultâneos por outros processos ou threads.

‘Fluxo’ do Programa

1. Recepção:

A cada novo cliente:

- Gera um tempo de paciência com base na prioridade.
- Escreve no arquivo `demanda.txt`.
- Adiciona o cliente na fila circular protegida por semáforos.

2. Atendimento:

- Retira o próximo cliente da fila.
- Calcula o tempo de atendimento e avalia a satisfação.
- Atualiza as métricas de desempenho.
- Escreve os dados do cliente no arquivo `LNG.txt`.

3. Analista:

- Acordado pelo atendente.
- Lê os registros do arquivo `LNG.txt`.
- Imprime no console e remove os registros processados.

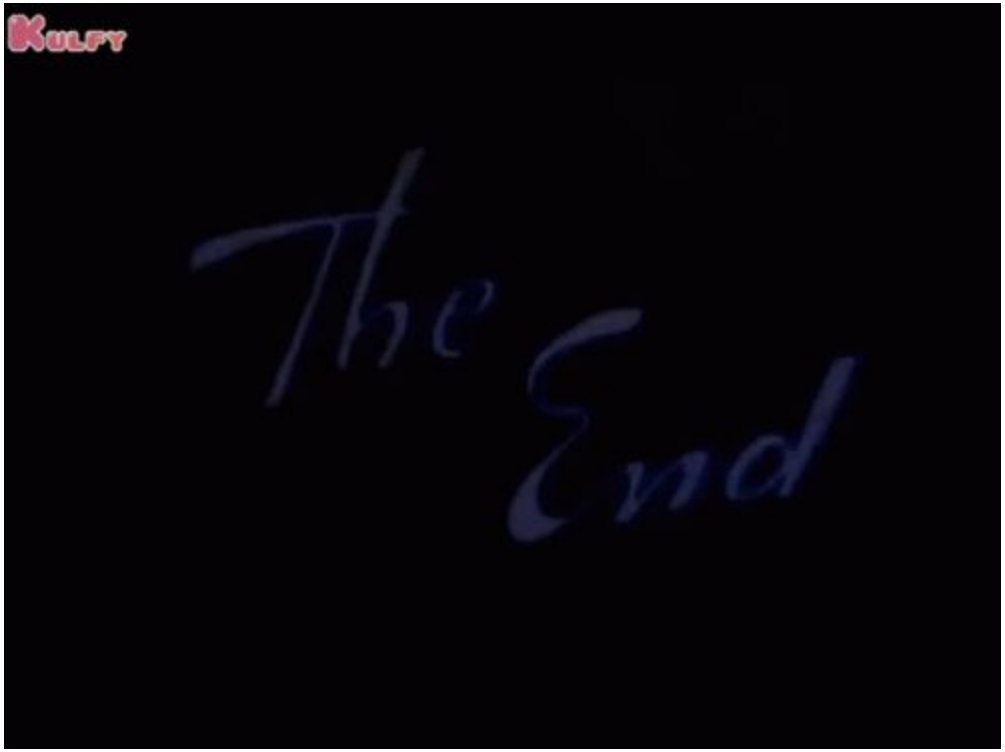
Dificuldades e Possíveis Melhorias

Dificuldades:

1. Gerenciamento do estado dos semáforos e organização com post e wait.
2. Mudar o fluxo do programa quando a fila fica cheia.
3. Amalgama dos processos, a ativação de paralisação dos semáforos está muito ligada ao funcionamento padrão

Melhorias:

1. Reduzir a quantidade de I/O nos arquivos de métricas e LNG.
2. Evitar bloqueios excessivos: O arquivo LNG.txt é bloqueado por toda a duração do processamento. Podia processar os dados em memória e atualizar só no final
3. Gerenciamento de semáforos residuais de execuções anteriores


KULFY
The End