UNIVERSIDADE DO VALE DO ITAJAÍ

Ciências da Computação
Disciplina: Sistemas Operacionais
Avaliação M1 – IPC, Threads e Paralelismo

Relatório de Implementação: Contador e Display para Esteiras de Produção

Professor Felipe Viel
Guilherme Rosa Dos Santos e Lucas Corrêa da Silva
Itajaí, 08 de abril de 2024

1. Enunciado do Projeto:

O problema consiste em implementar uma solução que realize a contagem de produtos em três esteiras de produção e exiba o resultado total em um display. Deve-se simular o sistema em um ambiente multicore com suporte a threads e IPC entre processos com pipes. Um processo será responsável pela contagem utilizando threads, enquanto outro processo será responsável pela apresentação no display.

2. Explicação e Contexto da Aplicação:

A solução proposta visa atender às necessidades de uma empresa que opera um sistema de produção com três esteiras, cada uma processando produtos de diferentes pesos. A contagem precisa ser precisa e atualizada em tempo real, enquanto os operadores precisam acompanhar os resultados em um display. A implementação utiliza threads para simular o funcionamento das esteiras e IPC por meio de pipes para comunicação entre os processos.

3. Resultados Obtidos com as Simulações:

Os resultados obtidos demonstram que a solução é capaz de contar os produtos de forma eficiente, atualizar o peso total e exibir os resultados no display conforme necessário. A comunicação entre os processos por meio de pipes garante uma integração adequada e a utilização de threads permite uma execução paralela e otimizada.

4. Códigos Importantes da Implementação:

A seguir, destacamos os trechos de código mais relevantes da implementação:

4.1 Display:

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

int main() {
    int file;                                           // Descritor do arquivo
    char * pipefile = "/tmp/pipefile";                  // Seleciona o arquivo que será usado para a comunicação entre os processos
    int bytesRead;

    int contagem;
    float pesoTotal;
    float ultimoPeso = 0;

    file = open(pipefile, O_RDONLY);                    // Abre o arquivo para leitura

    while(1) {
        bytesRead = read(file, &contagem, sizeof(contagem));    // Lê o arquivo e armazena o valor na variável contagem
        if(bytesRead != 0) {
            printf("Contagem: %d\n", contagem);                 // Exibe o valor da contagem
        }

        bytesRead = read(file, &pesoTotal, sizeof(pesoTotal));  // Lê o arquivo e armazena o valor na variável pesoTotal
        if(bytesRead != 0) {
            printf("Peso total: %.1fkg\n", pesoTotal);          // Exibe o valor do peso total
        }

        sleep(1);
    }

    close(file);                                        // Fecha o descritor do arquivo
    unlink(pipefile);                                   // Deleta o pipe

    return 0;
}
```

Essa parte do código representa o processo responsável pela exibição dos resultados da contagem e peso total em tempo real. Ele continuamente lê os dados do pipe e os exibe no terminal enquanto o programa estiver em execução.

Declaração de variáveis:

- “file”: Variável que armazena o descritor do arquivo;
- “pipefile”: Caminho do arquivo usado para a comunicação entre os processos por meio do pipe;
- “bytesRead”: Variável para armazenar o número de bytes lidos.

Abertura do arquivo de pipe:

Utiliza a função “open” para abrir o arquivo especificado em “pipefile” para leitura (O_RDONLY).

O descritor de arquivo resultante é armazenado na variável “file”.

Loop principal:

Um loop infinito (while(1)) é iniciado para ler continuamente os dados do pipe e exibi-los.

Dentro do loop:

Utiliza a função “read” para ler os dados do pipe e armazená-los nas variáveis “contagem” e “pesoTotal”.

Verifica se a leitura foi bem-sucedida (bytesRead != 0) antes de exibir os valores lidos.

Exibe a contagem e o peso total no terminal utilizando `printf`.

Aguarda 1 segundo entre cada leitura e exibição dos dados utilizando a função “sleep(1)”.

Fechamento do arquivo e remoção do pipe:

Após sair do loop, o programa fecha o descritor do arquivo usando a função “close”.

Em seguida, remove o pipe utilizando a função “unlink”.

4.2 Contagem:

Inicialização das Variáveis e Estruturas:

```
9     float pesos[1500];                                    // Vetor que armazena os pesos
10    int contagem = 0;
11    int contagemTotal = 0;
12    int running = 1;
13
14    pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;    // Inicializa o mutex
15    pthread_cond_t cond = PTHREAD_COND_INITIALIZER;       // Inicializa a variável de condição
16
17    typedef struct {
18        float peso;
19        int sleepTime;
20    } EsteiraInfo;
```

Essa parte do código inicializa variáveis cruciais para o funcionamento do sistema, como o vetor pesos para armazenar os pesos dos itens, variáveis para contagem total e controle do sistema, além de mutexes e variáveis de condição para garantir a sincronização entre threads. A estrutura “EsteiraInfo” armazena informações sobre o peso e o tempo de espera das esteiras.

Função “EsteiraThread”:

```
22  void* EsteiraThread(void* arg) {                    // Thread que simula as esteiras
23      EsteiraInfo* info = (EsteiraInfo*) arg;
24      while (1){
25          pthread_mutex_lock(&mutex);                 // Locka as variáveis compartilhadas com o mutex para evitar condição de corrida
26          while (!running) {
27              pthread_cond_wait(&cond, &mutex);       // Coloca a thread em espera até que a variável de condição seja satisfeita e libera o mutex
28          }
29          if (contagem < 1500){
30              pesos[contagem] = info->peso;
31              contagem++;
32              contagemTotal++;
33          }
34          pthread_mutex_unlock(&mutex);               // Desbloqueia o mutex
35          usleep(info->sleepTime);
36      }
37      return NULL;
38  }
```

Esta função representa o comportamento de cada esteira de produção em uma thread separada.

Ela aguarda até que o sistema esteja em execução (running == 1), utilizando um mutex, além de uma variável de condição para garantir uma execução sincronizada.

Se a contagem ainda não atingiu o limite de 1500 itens, ela atualiza o vetor pesos com o peso do item atual e incrementa a contagem.

Função “InputThread”:

```
40  void* InputThread(void* arg) {                      // Thread que lê a entrada do usuário
41      char input[64];
42      while (1) {
43          fgets(input, 64, stdin);
44          pthread_mutex_lock(&mutex);
45          if (strcmp(input, "pause\n") == 0) {
46              running = 0;
47          } else if (strcmp(input, "resume\n") == 0) {
48              running = 1;
49              pthread_cond_broadcast(&cond);          // Acorda todas as threads que estão esperando na variável de condição
50          }
51          pthread_mutex_unlock(&mutex);
52      }
53      return NULL;
54  }
```

Esta função é responsável por lidar com a entrada do usuário para pausar ou retomar a contagem.

Ela utiliza um loop infinito para aguardar a entrada do usuário e, dependendo da entrada, define o valor da variável running para pausar ou retomar a contagem. Em seguida, ela sinaliza todas as threads em espera usando pthread_cond_broadcast.

Loop Principal:

```
93        while (1) {
94            pthread_mutex_lock(&mutex);
95            if (running) {
96                write(file, &contagemTotal, sizeof(contagemTotal)); // Escreve a contagem no arquivo
97
98                if (contagem >= 1500){                          // Se a contagem for maior ou igual a 1500, atualiza o peso e reseta a contagem
99                    pararThreads();
100                   atualizarPeso(&pesoAtual);
101                   resetarContagem();
102                   retornarThreads();
103               }
104               write(file, &pesoAtual, sizeof(pesoAtual));         // Escreve o peso total no arquivo
105           }
106           pthread_mutex_unlock(&mutex);
107           sleep(1);
108       }
```

O loop principal é responsável por controlar o fluxo principal do programa.

Ele verifica se o sistema está em execução (running == 1) e, se estiver, escreve a contagem total e o peso atual no pipe para comunicação com o processo de exibição.

Quando a contagem atinge 1500 itens, as threads são pausadas, o peso é atualizado, a contagem é reiniciada e as threads são retomadas.

5. Análise e Discussão sobre os Resultados Finais:

A solução atendeu aos requisitos especificados pela empresa, oferecendo uma solução robusta e eficiente para o problema das três esteiras de produção. A utilização de threads e IPC por pipes demonstrou-se adequada para garantir uma execução concorrente segura e uma comunicação eficiente entre os processos. Possíveis melhorias podem ser exploradas para otimizar ainda mais o desempenho e a escalabilidade do sistema.