

Universidade Presbiteriana Mackenzie
Ciência da Computação – 05P11
Computação Paralela
20/02/2025

Alan Meniuk Gleizer
RA 10416804

Atividade – Monitoramento de Processos

Código Fonte

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int lerInt() {
    // funcao da biblioteca propria para ler um int sem os problemas de scanf
    char buffer[32]; // buffer para input
    int num;

    if (fgets(buffer, sizeof(buffer), stdin) == NULL) {
        return 0; // se fgets é NULL, houve erro de leitura ou EOF
    }

    if (sscanf(buffer, "%d", &num) != 1) {
        return 0; // verificar se entrada realmente é int
    }

    return num;
}

int main() {
    int totalFilhos = 0;
    do {
        printf("Informe a quantidade de filhos: ");
        totalFilhos = lerInt();
    } while (totalFilhos < 1);

    int numFilhos = 0;

    // Criar todos os filhos primeiro
    for (int i = 0; i < totalFilhos; i++) {
        pid_t pid = fork();
```

```

    if (pid < 0) {
        printf("Erro ao criar processo filho");
        return (-1);
    }

    if (pid == 0) {
        // no processo filho
        srand(time(NULL) + getpid());
        int sleep_time = rand() % 10 + 1;
        sleep(sleep_time);
        return (sleep_time);
    }
}

// aqui estamos no processo pai, pois todos os filhos deram return acima
// precisamos de dois loops! no primeiro fazemos os fork. aqui, esperamos todos
os filhos terminarem
for (int i = 0; i < totalFilhos; i++) {
    int status;
    pid_t pidFilho = wait(&status);
    /*
    wait() bloqueia execução do pai até que QUALQUER filho termine
    retorno de wait é o PID do filho que terminou.. por isso não precisamos
armazenar o PID
    o SO gerencia isso automaticamente
    */

    // apos cada filho terminar, pidFilho tem o pid, e WEXITSTATUS nos da o valor
de retorno

    if (WIFEXITED(status)) {
        printf("No processo pai: processo filho: %d, status de saída do filho:
%d\n", pidFilho, WEXITSTATUS(status));
    } else {
        printf("No processo pai: processo filho: %d terminou anormalmente.\n",
pidFilho);
    }
}

return 0;
}

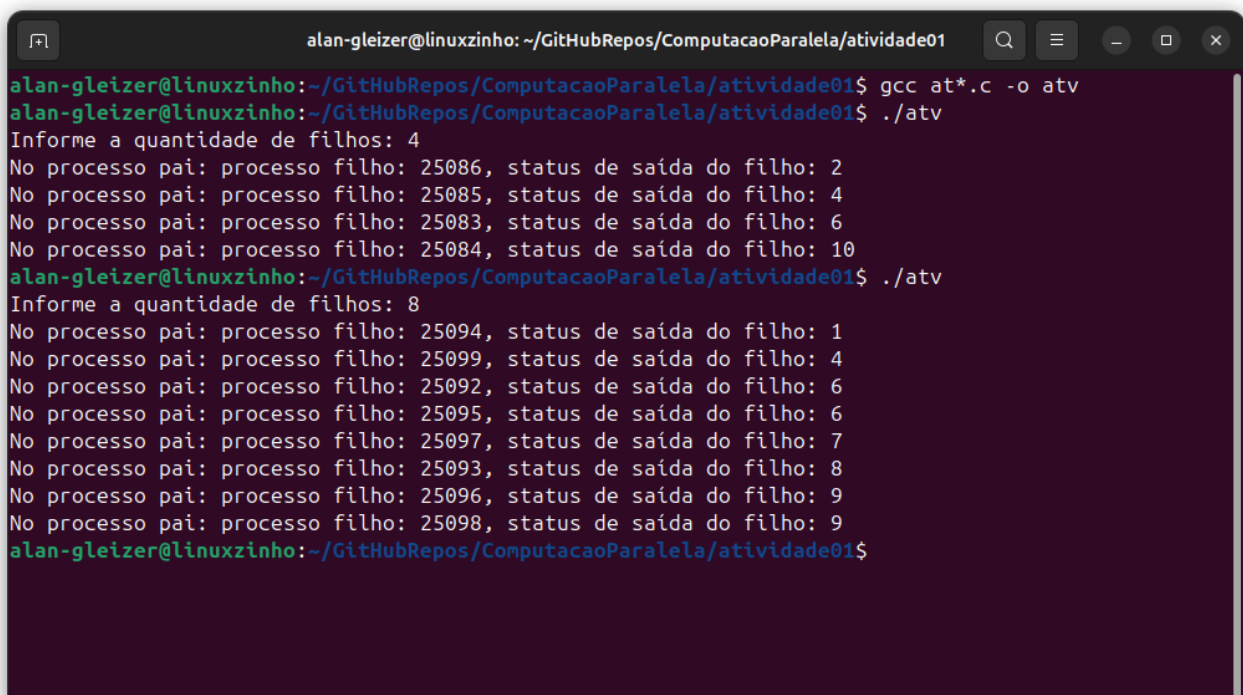
```

Comentários

O uso de `fork()` e `wait()`, junto com algumas das funções relacionadas como `WIFEXITED`, é desafiador na medida que o funcionamento dessas funções não é intuitivo em múltiplos sentidos. Em primeiro lugar, o desenvolvimento de um programa com execução de processos paralelos requer uma nova abordagem que evidentemente requer bastante prática. Na primeira tentativa, havia apenas um loop para `fork()`, `sleep()` e `wait()`. Isto é, um único loop no qual os processos filhos eram criados, executados, e no qual o pai esperava a execução terminar para imprimir o retorno do filho. Foi difícil perceber o erro: nesse caso, a execução de todos os filhos era sequencial, sem nenhum tipo de paralelismo. Também é necessário ter em mente quais partes do código serão executadas por pai ou filho, e até que ponto isso deve ocorrer.

Em segundo lugar, o funcionamento de muitas das funções citadas é um tanto obscuro, e requer consultas frequentes à documentação. Nos exemplos estudados em sala de aula, e no código desenvolvido, `wait()` recebe como parâmetro `&status`, mas a variável `status` é declarada e nunca é explicitamente alterada. Pela documentação, ficou claro que o conteúdo de `status` é alterado e gerenciado pelo kernel. Da mesma forma, o funcionamento de `WIFEXITED` e funções alternativas requer familiaridade com o SO e as bibliotecas associadas.

Print de Execução

A terminal window with a dark background and light green text. The window title is 'alan-gleizer@linuxzinho: ~/GitHubRepos/ComputacaoParalela/atividade01'. The user enters 'gcc at*.c -o atv' and './atv'. The program prompts 'Informe a quantidade de filhos: 4' and then prints four lines of status for each child process. The user then enters './atv' again, the program prompts 'Informe a quantidade de filhos: 8', and prints eight lines of status for each child process.

```
alan-gleizer@linuxzinho: ~/GitHubRepos/ComputacaoParalela/atividade01
alan-gleizer@linuxzinho:~/GitHubRepos/ComputacaoParalela/atividade01$ gcc at*.c -o atv
alan-gleizer@linuxzinho:~/GitHubRepos/ComputacaoParalela/atividade01$ ./atv
Informe a quantidade de filhos: 4
No processo pai: processo filho: 25086, status de saída do filho: 2
No processo pai: processo filho: 25085, status de saída do filho: 4
No processo pai: processo filho: 25083, status de saída do filho: 6
No processo pai: processo filho: 25084, status de saída do filho: 10
alan-gleizer@linuxzinho:~/GitHubRepos/ComputacaoParalela/atividade01$ ./atv
Informe a quantidade de filhos: 8
No processo pai: processo filho: 25094, status de saída do filho: 1
No processo pai: processo filho: 25099, status de saída do filho: 4
No processo pai: processo filho: 25092, status de saída do filho: 6
No processo pai: processo filho: 25095, status de saída do filho: 6
No processo pai: processo filho: 25097, status de saída do filho: 7
No processo pai: processo filho: 25093, status de saída do filho: 8
No processo pai: processo filho: 25096, status de saída do filho: 9
No processo pai: processo filho: 25098, status de saída do filho: 9
alan-gleizer@linuxzinho:~/GitHubRepos/ComputacaoParalela/atividade01$
```