Monitoramento de Processos Filhos

Mário O. de Menezes
Faculdade de Computação e Informática – FCI
Universidade Presbiteriana Mackenzie

Monitorando Processos Filhos

Como vimos, sempre que criamos um processo filho a partir de um programa usando **fork**, o seguinte acontece:

- O processo atual torna-se o processo pai.
- O novo processo torna-se o processo filho.

O que acontece se o processo pai concluir sua tarefa antes do processo filho e depois encerrar ou sair? Agora, quem seria o pai do processo filho?

O pai do processo filho é o processo `init`, que é o primeiro processo a iniciar todas as tarefas.

Para monitorar o estado de execução do processo filho, verificar se o processo filho está em execução, parado ou para verificar o status de execução, etc., são utilizadas as chamadas de sistema `wait()` e suas variantes.

A chamada `wait()`

Primeiro Exemplo

Vamos considerar um programa de exemplo, onde o processo pai não espera pelo processo filho, o que resulta no processo `init` se tornando o novo pai do processo filho.

Nome do arquivo: `parentprocess_nowait.c`

```c
#include <stdio.h>

int main() {
  int pid;
  pid = fork();

  // Processo filho
  if (pid == 0) {
    system("ps -ef");
    sleep(10);
    system("ps -ef");
  } else {
    sleep(3);
  }
  return 0;
}
```

Variantes de `wait()`

Seguem as variantes das chamadas de sistema para monitorar o(s) processo(s) filho(s):

```
wait()
waitpid()
waitid()
```

A chamada de sistema `wait()` aguarda a terminação de um dos processos filhos e retorna seu status de terminação em um buffer, conforme explicado a seguir.

```c
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *status);
```

- Essa chamada retorna o ID do processo do filho que foi encerrado em caso de sucesso e -1 em caso de falha.
- A chamada de sistema `wait()` suspende a execução do processo atual e aguarda indefinidamente até que um de seus filhos seja encerrado.
- O status de terminação do filho fica disponível em status.

Segundo Exemplo

Vamos modificar o programa anterior para que o processo pai agora aguarde o processo filho.

Nome do arquivo: `parentprocess_waits.c`

```c
#include <stdio.h>

int main() {
  int pid;
  int status;
  pid = fork();

  // Processo filho
  if (pid == 0) {
    system("ps -ef");
    sleep(10);
    system("ps -ef");
    return 3;  // Status de saída é 3 do processo filho
  } else {
    sleep(3);
    wait(&status);
    printf("No processo pai: status de saída do filho é decimal %d, hexa %0x\n", status, status);
  }
  return 0;
}

```

Código de retorno de `wait()`

Observação:

- Embora o filho retorne o status de saída 3, o processo pai o vê como 768.
- O status é armazenado no byte de ordem superior, portanto, é armazenado em formato hexadecimal como 0X0300, que é 768 em decimal.

A terminação normal é a seguinte:

Byte de Ordem Superior (Bits 8 a 15)	Byte de Ordem Inferior (Bits 0 a 7)
Status de Saída (0 a 255)	0

- A chamada de sistema `wait()` possui a limitação de só poder esperar pela saída do próximo filho.
- Se precisarmos esperar por um filho específico, não é possível usando `wait()`, porém, é possível usando a chamada de sistema `waitpid()`.

A Chamada `waitpid()`

A chamada de sistema `waitpid()` aguarda a terminação de filhos especificados e retorna seu status de terminação no buffer, conforme explicado abaixo.

```c
#include <sys/types.h>
#include <sys/wait.h>

pid_t waitpid(pid_t pid, int *status, int options);
```

- A chamada acima retorna o ID do processo do filho que foi encerrado em caso de sucesso e -1 em caso de falha.
- A chamada de sistema `waitpid()` suspende a execução do processo atual e aguarda indefinidamente até que o filho especificado (conforme o valor de `pid`) termine.
- O status de terminação do filho fica disponível em status.

O valor de pid pode ser qualquer um dos seguintes:

- **< -1**: Aguardar por qualquer processo filho cujo ID de grupo de processo seja igual ao valor absoluto de pid.
- **-1**: Aguardar por qualquer processo filho, equivalente à chamada de sistema wait().
- **0**: Aguardar por qualquer processo filho cujo ID de grupo de processo seja igual ao do processo de chamada.
- **>0**: Aguardar por qualquer processo filho cujo ID de processo seja igual ao valor de pid.

Por padrão, a chamada de sistema `waitpid()` espera apenas por filhos terminados, mas esse comportamento padrão pode ser modificado usando o argumento options.

Terceiro Exemplo

Agora, vamos considerar um programa como exemplo, esperando por um processo específico com seu ID de processo.

Nome do arquivo: `waitpid_test.c`

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

int main() {
  int pid;
  int pids[3];
  int status;
  int num_processos = 0;
  int total_processos = 3;

  while (num_processos < total_processos) {
    pid = fork();

    // Processo filho
    if (pid == 0) {
      printf("No processo filho: ID do processo é %d\n", getpid());
      sleep(5);
      return 4;
    } else {
      pids[num_processos] = pid;
      num_processos++;
      printf("No processo pai: processo número %d criado\n", pid);
    }
  }

  // Aguardando o 3° processo filho
  waitpid(pids[total_processos - 1], &status, 0);
  if (WIFEXITED(status) != 0) {
    printf("Processo %d encerrado normalmente\n", pids[total_processos - 1]); (1)
    printf("Status de saída do filho é %d\n", WEXITSTATUS(status)); (2)
  } else {
```

Resultado do Terceiro Exemplo

Após a compilação e execução, a saída é a seguinte:

```
No processo filho: ID do processo é 32528
No processo pai: processo número 32528 criado
No processo filho: ID do processo é 32529
No processo pai: processo número 32528 criado
No processo pai: processo número 32529 criado
No processo filho: ID do processo é 32530
No processo pai: processo número 32528 criado
No processo pai: processo número 32529 criado
No processo pai: processo número 32530 criado
Processo 32530 encerrado normalmente
Status de saída do filho é 4
```

A chamada `waitid()`

Agora, vamos analisar a chamada de sistema `waitid()`. Esta chamada de sistema aguarda a mudança de estado do processo filho.

```
1 #include <sys/wait.h>
2
3 int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
```

- A chamada de sistema acima aguarda a mudança de estado do processo filho e suspende o processo atual/chamado até que qualquer um de seus processos filhos mude seu estado.
- O argumento infop serve para registrar o estado atual do filho.
- Esta chamada retorna imediatamente, se o processo já mudou de estado.

A Chamada `waitid()` (cont.)

O valor de `idtype` pode ser qualquer um dos seguintes:

- **P_PID**: Aguarda qualquer processo filho cujo ID do processo seja igual ao de id.
- **P_PGID**: Aguarda qualquer processo filho cujo ID do grupo de processos seja igual ao de id.
- **P_ALL**: Aguarda qualquer processo filho e id é ignorado.

O argumento options é usado para especificar quais mudanças de estado e pode ser formado com a operação `OU` bit a bit com os seguintes flags:

- `WCONTINUED`: Retorna o status de qualquer filho que foi parado e foi continuado.
- `WEXITED`: Aguarda o processo finalizar.
- `WNOHANG`: Retorna imediatamente.
- `WSTOPPED`: Aguarda o processo de qualquer filho que tenha parado ao receber o sinal e retorna o status.

Esta chamada retorna 0, se retornar devido a uma mudança de estado de um de seus filhos e `WNOHANG` for usado.

Retorna -1, em caso de erro e define o número de erro apropriado.

Quarto Exemplo

Nome do arquivo: `waitid_test.c`

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

int main() {
  int pid;
  int pids[3];
  int status;
  int num_processos = 0;
  int total_processos = 3;
  siginfo_t siginfo;

  while (num_processos < total_processos) {
    pid = fork();

    // Processo filho
    if (pid == 0) {
      printf("No processo filho: ID do processo é %d\n", getpid());
      sleep(5);
      return 4;
    } else {
      pids[num_processos] = pid;
      num_processos++;
      printf("No processo pai: processo número %d criado\n", pid);
    }
```

Resultado do Quarto Exemplo

Após a execução e compilação do programa acima, o seguinte resultado é obtido:

```
No processo filho: ID do processo é 35390
No processo pai: processo número 35390 criado
No processo filho: ID do processo é 35391
No processo pai: processo número 35390 criado
No processo pai: processo número 35391 criado
No processo filho: ID do processo é 35392
No processo pai: processo número 35390 criado
No processo pai: processo número 35391 criado
No processo pai: processo número 35392 criado
Informações recebidas do waitid são: PID do filho: 35392,
  ID de usuário real do filho: 4581875
```

Exercícios

1. Implemente um programa utilizando a chamada de função `wait()` que espere todos os `n` processos filhos terminarem, e imprima na tela o **PID** e o código de retorno de cada um. O processo *pai* **não** deve guardar os **PID**s dos filhos criados, e deve mostrar, no mínimo, a seguinte saída:

```
$ ./exercicio1
No processo pai: processo filho: 64464, status de saída do filho: 1
No processo pai: processo filho: 64463, status de saída do filho: 3
No processo pai: processo filho: 64462, status de saída do filho: 8
No processo pai: processo filho: 64465, status de saída do filho: 10
```

O código de retorno dos filhos deve ser o tempo que ficam em `sleep`; você pode utilizar um tempo aleatório para o `sleep` da seguinte forma:

```
#include <stdlib.h>
#include <time.h>
...
srand(time(NULL)+getpid());
sleep_time = rand() % 10 + 1;
sleep(sleep_time);
...
```

2. Substitua a chamada `wait()` por `waitpid()` no programa anterior, mantendo as demais funcionalidades.