Universidade Federal do Pará
Instituto de Ciências Exatas e Naturais
Faculdade de Computação
Laboratório de Sistemas Operacionais

Experiência 04 – Criação de processos em Linux.

A chamada de sistema *fork()* é utilizada para criar processos em linguagem C. Não é necessário nenhum argumento e o valor de retorno é o identificador (ID) do processo criado.

O propósito de *fork()* é criar um novo processo que se torna ”filho” do processo que o chamou e após a criação do processo, o processo “pai” e “filho” irão executar a próxima instrução após a chamada de *fork()*.

A chamada de sistema *fork()* faz uma cópia exata do espaço de endereçamento do processo “pai” para o processo “filho”, ou seja, os processos possuem espaços de endereços separados, fazendo com que alterações no processo “pai” não tenham repercussão no processo “filho” e vice-versa.

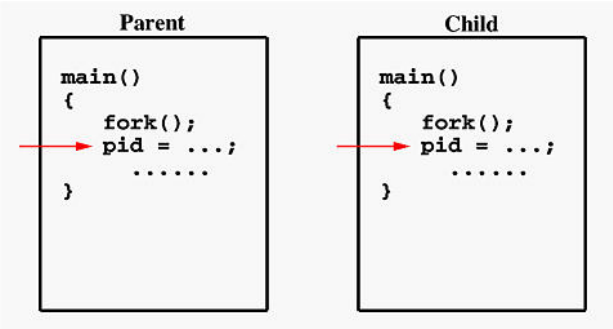
O programa abaixo cria um processo “pai” e um processo “filho” com a chamada de sistema *fork()*.

Programa 01 – Criação de processo ”pai” e processo “filho”

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

#define   MAX_COUNT  200
#define   BUF_SIZE   100

void  main(void)
{
    pid_t  pid;
    int    i;
    char   buf[BUF_SIZE];

    fork();
    pid = getpid();
    for (i = 1; i <= MAX_COUNT; i++) {
        sprintf(buf, "PID = %d, valor = %d\n", pid, i);
        write(1, buf, strlen(buf));
    }

    exit(0);
}
```

Após a chamada de *fork()* no programa acima, o sistema terá 02 (dois) processos em execução.

Figura 01 – Criação de 2 processos com *fork()*.



O programa abaixo cria os mesmos processos acima, mas faz distinção na utilização de 02 (duas) funções para cada um deles.

Programa 02 - Criação de processo ”pai” e processo “filho” com funções separadas.

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

#define   MAX_COUNT  200

void  ChildProcess(void);
void  ParentProcess(void);

void  main(void) {
    pid_t  pid;

    pid = fork();
    if (pid == 0)
        ChildProcess();
    else
        ParentProcess();

    exit(0);
}

void  ChildProcess(void) {
    int   i;

    for (i = 1; i <= MAX_COUNT; i++)
        printf("   Processo FILHO, valor = %d\n", i);
    printf("   *** Processo FILHO finalizado ***\n");
}

void  ParentProcess(void) {
    int   i;

    for (i = 1; i <= MAX_COUNT; i++)
        printf("Processo PAI, value = %d\n", i);
    printf("*** Processo PAI finalizado ***\n");

    exit(0);

}
```

A figura 02 mostra a criação de processos e identificação deles através do identificador do processo retornado pela chamada de sistema *fork()*. Quando a chamada de sistema *fork()* retorna um valor diferente de 0 (zero) significa que o processo em execução é o pai. Caso o valor retornado por *fork()* seja 0 (zero), o processo em execução é o filho. As figuras 03 e 04 mostram a tomada de decisão em função do valor da variável *pid* que possui o identificador do processo (PID).

Figura 02 – Criação de 2 processos com *fork()* organizados em funções.

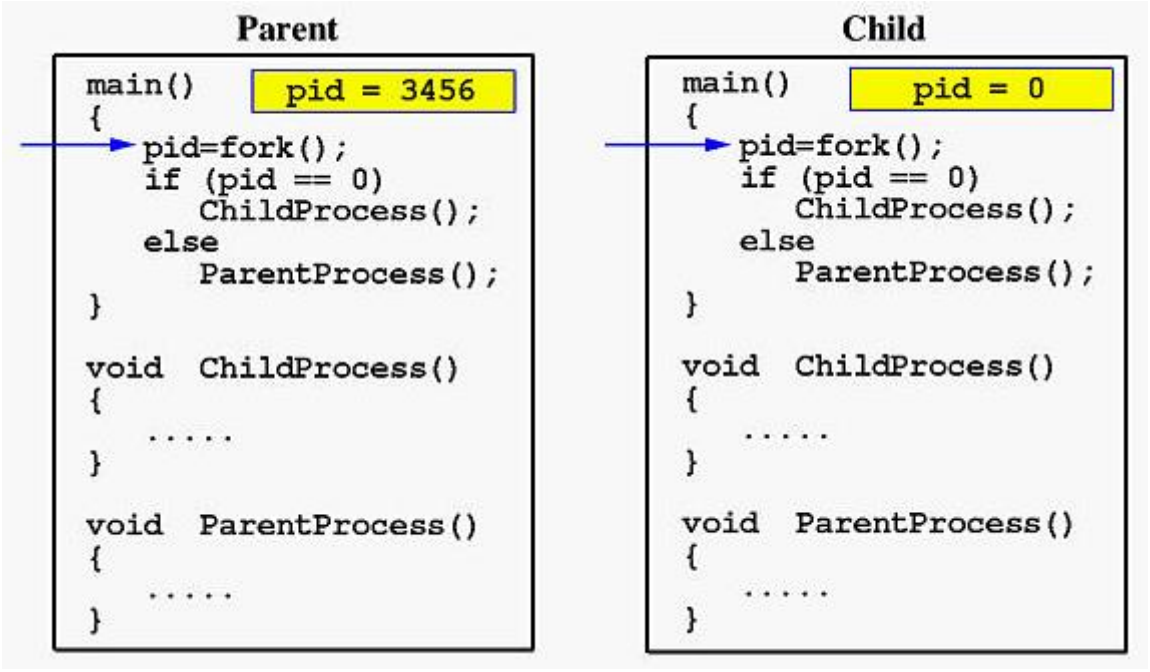


Figura 02 – Tomada de decisão em função do valor retornado por *fork()*.

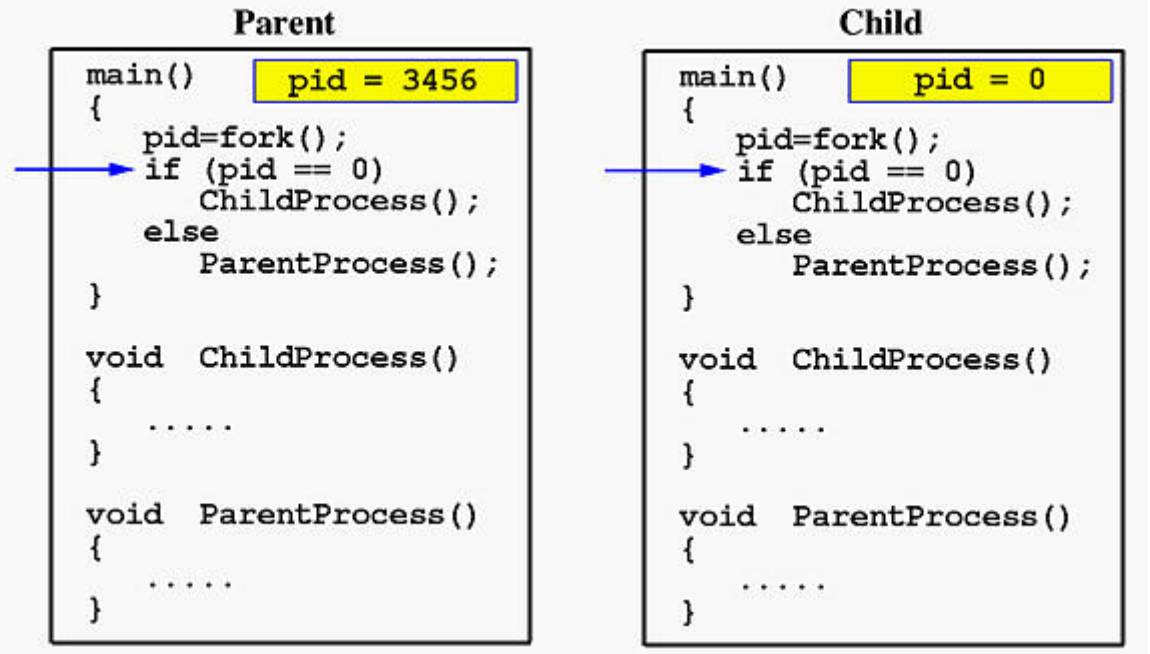
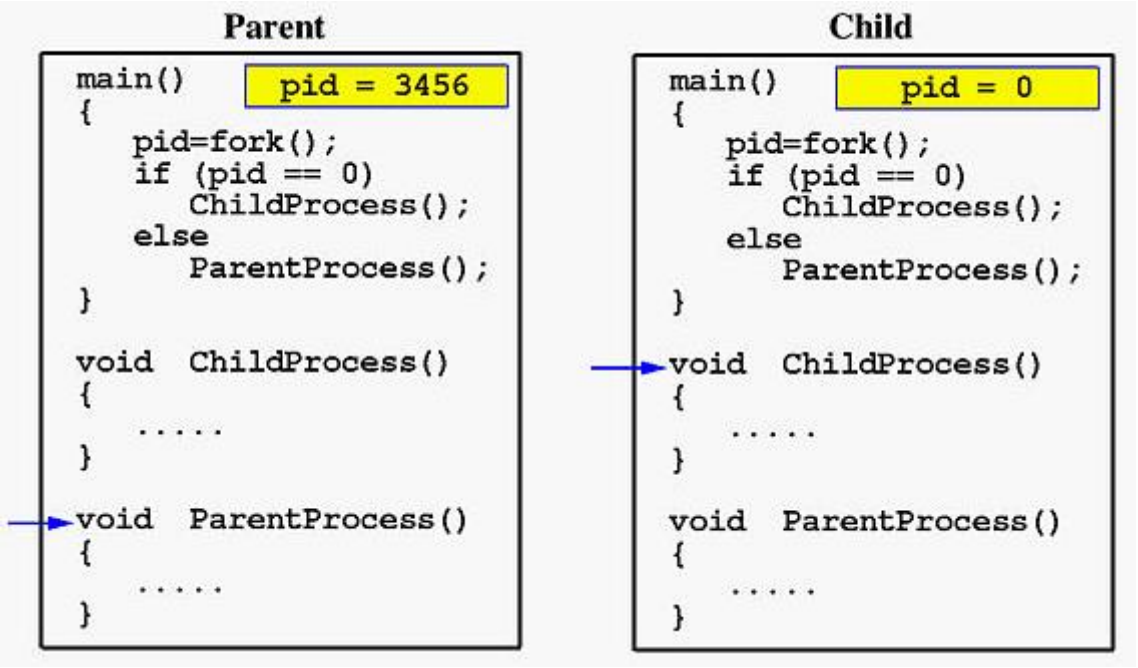


Figura 02 – Execução após tomada de decisão em função do valor retornado por *fork()*.



TAREFA 01

a) A CPU separa uma fatia de tempo para execução de processos. Caso o processo necessite de um tempo maior que essa fatia de tempo, o processo é retirado da CPU e colocado em que estado?
b) Em relação a fatia de tempo, qual o comportamento do programa anterior?
c) Como fazer para que o programa anterior tenha um comportamento diferente do citado acima?
d) Verifique o aumento de iterações que o programa deve fazer para que ele ocupe mais de uma vez a CPU.

TAREFA 02

Crie um programa que execute 03 tarefas diferentes, conforme colocado abaixo.

a) Cálculo do fatorial de um número fixo < 20;
b) Mostrar os 100 primeiros números da série de Fibonacci;
c) Executar o algoritmo da Torre de Hanoi para 06 (discos).

A execução dos processos deve ser efetuada da seguinte forma: pai -> filho -> neto.