Invoke child and parent process engineering essay

Engineering

Student’s NameReg. No. Date of submission: C. W. S. Goonetilleke: 409064899: 04/03/2013AIM: Write a C program using the fork () system call that generates the Fibonacci sequence in the child process in Linux. OBJECTIVES: - Familiar with process.- Invoke child and parent process.- Familiar with Linux and windows system calls.

THEORY:

Process in operating systems

A running instance of a program can be identified as a process. It includes the current activities, states of the program execution and several information that belongs to program execution. When process execute, executable file is loaded into memory area that normally called a process address space.

Child process and a parent process

Parent process is the process that create another process and the created process by parent process is named as the child process. Child process inherits most of parent process’s attributes, such as file descriptors. Each process may create more than one child processes. But child process may only has single parent process. Process does not have a parent, when it was created directly by the kernel. When parent process terminated, its child process will be leaving as an orphan process. But after child process become orphan it will be adopted by the kernel process.

Structure of a process in memory after an executable file is loaded in to memory

After loading process into memory its address space is divided into three segments as follows. Text segment: the area in which the executable or binary image instructions reside. Data segment: the area which statically allocated and global data. Stack segment: the area where local variables are allocated.

Process states changing

http://www. d. umn. edu/~gshute/os/images/process-diagram. pngAfter creation of process it admitted into the ready state. Ready state process has all of the resources that it needs for further execution. Ready state process is normally held in a ready queue until a processor becomes available. When processor become available process dispatch. Then it became to running state and execute its instructions. If any time out occurred process again back to Ready state. Anyway if required resource blocked by another process or process needs to response from resource, process become to block state. It still hold in block state until the required resource available. Afterwards process become Ready state again and waiting for processor becomes available. After completing instruction process terminated and release resources if holds any.

Process Control Block (PCB)

PCB for a process contains resource management information, administrative information and execution snapshot. Resource management information includes information about required resource such as open files, page tables and I/O streams. Administrative information includes a process

identifier, resource usage information such as execution time, process ownership, and administrative data needed for system security. The execution snapshot captures information such as register contents and Program Counter states that is required to restore its operation back.

PCB in a Linux operating system

http://4. bp. blogspot. com/_cga7sfO2vms/TOsAHnl174I/AAAAAAAAElE/qijm_yA6vqM/s1600/PCB. pngPointer: to Next process and to Previous processProcess state: Creation state such as unrunnable, runnable or terminatedRunnable state such as ready, block or runningPriority stateProcess number: Error number or exit codeProgram counter: Instruction pointer of loaded instruction in processRegisters: Flag registers, debug registersMemory limits: Memory domain of processList of open filesThose various data are struct by Linux kernel as follows, volatile long state; /* -1 unrunnable, 0 runnable, > 0 stopped */long counter; long priority; unsigned long signal; unsigned long blocked; /* bitmap of masked signals */unsigned long flags; /* per process flags, defined below */int errno; long debugreg[8]; /* Hardware debugging registers */struct exec_domain *exec_domain;/* various fields */struct linux_binfmt *binfmt; struct task_struct *next_task, *prev_task; struct task_struct *next_run, *prev_run; unsigned long saved_kernel_stack; unsigned long kernel_stack_page; int exit_code, exit_signal;

TASK 1

PROCEDURE:

Use gedit and code c program for create sub process using foke() system call and wait parent process until child process being completed using wait() system call in Linux. Write program to find Fibonacci sequence for given number. Save code and compile it using gcc. Execute compiled program.#include void main(){long int fib0 = 0, fib1 = 1, fibn; int n; printf(" Number of Fibonacci Sequence: "); scanf("%d", &n); if (n >= 0){pid_t pid = fork(); if (pid == 0){printf(" Child process of PID:%d is going to work on Fibonacci Sequence", getppid()); long int a = fib0, b = fib1; int i; printf("%d, %d, ", fib0, fib1); if (n > 1){n = n - 1; fibn= a+b; printf("%d, ", fibn); a= b; b= fibn;

}

}

printf(" Child process (PID:%d) ends", getpid());}else if (pid > 0){printf(" Parent process (PID:%d) is waiting for child process (PID:%d) to complete", getpid(), pid); wait(pid); printf(" Parent process (PID:%d) ends", getpid());}else{printf(" Ooops.. Something goes wrong!");

}

}else {printf(" Try again.. You must enter positive integer as a number");

}

}

OBSERVATIONS AND RESULTS:

TASK 2

PROCEDURE:

Create c++ console solution using visual studio. Create project under solution for child process. Code child process as follows to find Fibonacci sequence of a number.#include void main(){long int fib0 = 0, fib1 = 1, fibn; int n; DWORD pid = GetCurrentProcessId(); printf(" Child process of PID:%d is going to work on Fibonacci Sequence", pid); printf(" Number of Fibonacci Sequence: "); scanf_s("%d", &n); if (n >= 0){long int a = fib0, b = fib1; int i; printf("%d, %d, ", fib0, fib1); if (n > 1){n = n - 1; fibn= a+b; printf("%d, ", fibn); a= b; b= fibn;

}

}

printf(" Child process (PID:%d) ends", pid);}else {printf(" Try again.. You must enter positive integer as a number");

}

}

Compile and Run program to verify its functionality. Create project under solution for parent process andSet solution properties to build child first and then build parent process. Set debugging project as parent project. Write program for parent process as follow to get child process as input parameter of parent process and to execute it via parent process using createProcess()

system call in win 32 API.#include #include #include void _tmain(int argc, TCHAR *argv[])

{

STARTUPINFO si; PROCESS_INFORMATION pi; ZeroMemory(&si, sizeof(si)); si. cb = sizeof(si); ZeroMemory(π, sizeof(pi)); DWORD pid = GetCurrentProcessId(); if(argc != 2)

{

printf(" Usage: task2 [child process name]", argv[0]); return;

}

printf(" Parent Process (PID:%d) Started.", pid);// Start the child process. if(! CreateProcess(NULL, // No module nameargv[1], // Command lineNULL, // Process handle not inheritableNULL, // Thread handle not inheritableTRUE, // Set handle inheritance to TRUE0, // No creation flagsNULL, // Use parent's environment blockNULL, // Use parent's starting directory&si, // Pointer to STARTUPINFO structureπ) // Pointer to PROCESS_INFORMATION structure

)

{

printf(" CreateProcess failed (%d).", GetLastError()); return;

}

// Wait until child process exits. printf(" Parent Process (PID:%d) Wait until child process exits", pid); WaitForSingleObject(pi. hProcess, INFINITE);// Close process and thread handles. CloseHandle(pi. hProcess);

```
CloseHandle( pi. hThread ); printf( " Parent Process (PID:%d)

CompleatedPress any key to exit", pid); _getch();

}
```

OBSERVATIONS AND RESULTS: