|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

*ФАКУЛЬТЕТ «Информатика и системы управления»*

*КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»*

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 04 |

**Название:**

***Процессы. Системные вызовы fork() и exec().***

**Дисциплина:  *Операционные системы***

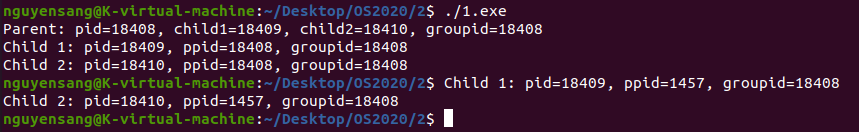
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ***ИУ7И-56Б*** |  |  | **Нгуен Ф. С.** |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | **Рязанова Н. Ю.** |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

*Москва, 2020*

* **Задание 1:**

Написать программу, запускающую не мене двух новых процессов системным вызовом fork(). В предке вывести собственный идентификатор (функция getpid()), идентификатор группы ( функция getpgrp()) и идентификаторы потомков. В процессе-потомке вывести собственный идентификатор, идентификатор предка (функция getppid()) и идентификатор группы. Убедиться, что при завершении процесса-предка потомок, который продолжает выполняться, получает идентификатор предка (PPID), равный 1 или идентификатор процесса-посредника.

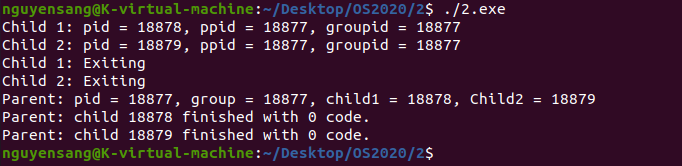
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <sys/types.h>
4. #include <unistd.h>
6. #define DELAY 4
8. **int** main() {
9. pid\_t child1 = fork();
11. **if** (child1 == -1)
12. {
13. perror("Can't fork Child 1");
14. exit(1);
15. }
16. **else** **if** (child1 == 0) // Child 1
17. {
18. printf("Child 1: pid=%d, ppid=%d, groupid=%d\n", getpid(), getppid(), getpgrp());
19. sleep(DELAY);
20. printf("Child 1: pid=%d, ppid=%d, groupid=%d\n", getpid(), getppid(), getpgrp());
21. **return** 0;
22. }
24. pid\_t child2 = fork();
25. **if** (child2 == -1)
26. {
27. perror("Can't fork");
28. exit(1);
29. }
30. **else** **if** (child2 == 0) // Child 2
31. {
32. printf("Child 2: pid=%d, ppid=%d, groupid=%d\n", getpid(), getppid(), getpgrp());
33. sleep(DELAY);
34. printf("Child 2: pid=%d, ppid=%d, groupid=%d\n", getpid(), getppid(), getpgrp());
35. **return** 0;
36. }
38. // Parent
39. printf("Parent: pid=%d, child1=%d, child2=%d, groupid=%d\n", getpid(), child1, child2, getpgrp());
40. **return** 0;
41. }



* **Задание 2:**

Написать программу по схеме первого задания, но в процессе-предке выполнить системный вызов wait(). Убедиться, что в этом случае идентификатор процесса потомка на 1 больше идентификатора процесса-предка.

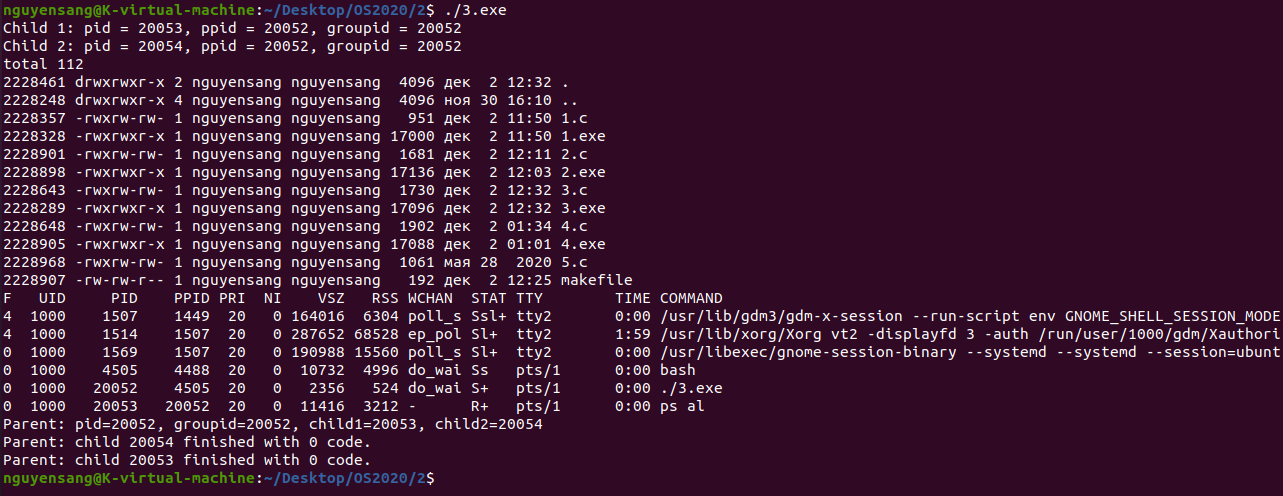
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <sys/types.h>
4. #include <sys/wait.h>
5. #include <unistd.h>
7. #define DELAY 5
9. **int** main() {
10. // Fork Child 1
11. pid\_t child1 = fork();
12. **if** (child1 == -1)
13. {
14. perror("Couldn't fork.");
15. exit(1);
16. }
17. **else** **if** (child1 == 0)
18. {
19. printf("Child 1: pid = %d, ppid = %d, groupid = %d\n", getpid(), getppid(), getpgrp() );
20. sleep(DELAY);
21. printf("Child 1: Exiting\n");
22. **return** 0;
23. }
25. // Fork Child 2
26. pid\_t child2 = fork();
27. **if** (child2 == -1)
28. {
29. perror("Couldn't fork.");
30. exit(1);
31. }
32. **else** **if** (child2 == 0)
33. {
34. printf("Child 2: pid = %d, ppid = %d, groupid = %d\n", getpid(), getppid(), getpgrp());
35. sleep(DELAY);
36. printf("Child 2: Exiting\n");
37. **return** 0;
38. }
40. // Parent
41. **int** status1;
42. pid\_t ret1 = wait(&status1);
43. **int** status2;
44. pid\_t ret2 = wait(&status2);
46. printf("Parent: pid = %d, group = %d, child1 = %d, Child2 = %d\n", getpid(), getpgrp(), child1, child2 );
48. **if** (WIFEXITED(status1))
49. printf("Parent: child %d finished with %d code.\n", ret1, WEXITSTATUS(status1));
50. **else** **if** (WIFSIGNALED(status1))
51. printf("Parent: child %d finished from signal with %d code.\n", ret1, WTERMSIG(status1));
52. **else** **if** (WIFSTOPPED(status1))
53. printf("Parent: child %d finished from signal with %d code.\n", ret1, WSTOPSIG(status1));
55. **if** (WIFEXITED(status2))
56. printf("Parent: child %d finished with %d code.\n", ret2, WEXITSTATUS(status2));
57. **else** **if** (WIFSIGNALED(status2))
58. printf("Parent: child %d finished from signal with %d code.\n", ret2, WTERMSIG(status2));
59. **else** **if** (WIFSTOPPED(status2))
60. printf("Parent: child %d finished from signal with %d code.\n", ret2, WSTOPSIG(status2));
62. **return** 0;
63. }



* **Задание 3:**

Написать программу, в которой процесс-потомок вызывает системный вызов exec(), а процесс-предок ждет завершения процесса-потомка. Следует создать не менее двух потомков.

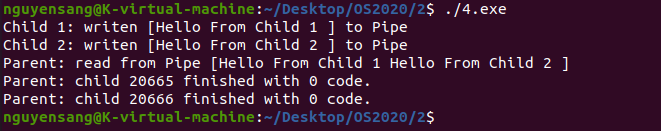
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <sys/types.h>
5. #include <sys/wait.h>
7. **int** main() {
8. // Child 1
9. pid\_t child1 = fork();
10. **if** (child1 == -1)
11. {
12. perror("Child 1: Couldn't fork.");
13. exit(1);
14. }
15. **else** **if** (child1 == 0)
16. {
17. printf("Child 1: pid = %d, ppid = %d, groupid = %d\n", getpid(), getppid(), getpgrp());
19. **if** (execlp("ps", "ps", "al", 0) == -1)
20. {
21. perror("Child 1: couldn't exec.");
22. exit(1);
23. }
24. }
25. // Child 2
26. pid\_t child2 = fork();
27. **if** (child2 == -1)
28. {
29. perror("Child 2: Couldn't fork.");
30. exit(1);
31. }
32. **else** **if** (child2 == 0)
33. {
34. printf("Child 2: pid = %d, ppid = %d, groupid = %d\n", getpid(), getppid(), getpgrp());
36. **if** (execlp("ls", "ls", "-ail", 0) == -1)
37. {
38. perror("Child 2: couldn't exec.");
39. exit(1);
40. }
41. }
42. // Parent
43. **if** (child1 != 0 && child2 != 0)
44. {
45. **int** status1, status2;
46. pid\_t ret1 = wait(&status1);
47. pid\_t ret2 = wait(&status2);
49. printf("Parent: pid=%d, groupid=%d, child1=%d, child2=%d\n", getpid(), getpgrp(), child1, child2);
51. **if** (WIFEXITED(status1))
52. printf("Parent: child %d finished with %d code.\n", ret1, WEXITSTATUS(status1));
53. **else** **if** (WIFSIGNALED(status1))
54. printf("Parent: child %d finished from signal with %d code.\n", ret1, WTERMSIG(status1));
55. **else** **if** (WIFSTOPPED(status1))
56. printf("Parent: child %d finished from signal with %d code.\n", ret1, WSTOPSIG(status1));
58. **if** (WIFEXITED(status2))
59. printf("Parent: child %d finished with %d code.\n", ret2, WEXITSTATUS(status2));
60. **else** **if** (WIFSIGNALED(status2))
61. printf("Parent: child %d finished from signal with %d code.\n", ret2, WTERMSIG(status2))
62. **else** **if** (WIFSTOPPED(status2))
63. printf("Parent: child %d finished from signal with %d code.\n", ret2, WSTOPSIG(status2));
64. }
66. **return** 0;
67. }



* **Задание 4:**

Написать программу, в которой предок и потомок обмениваются сообщением через программный канал.

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <string.h>
5. #include <sys/types.h>
6. #include <sys/wait.h>
8. #define LEN 100
10. **int** main() {
11. // Pipe
12. **int** fd[2];
13. **if** (pipe(fd) == -1)
14. {
15. perror("Couldn't pipe.");
16. exit(1);
17. }
18. // Child 1
19. pid\_t child1 = fork();
20. **if** (child1 == -1)
21. {
22. perror("Couldn't fork.");
23. exit(1);
24. }
25. **else** **if** (child1 == 0)
26. {
27. close(fd[0]);
28. **char** msg1[] = "Hello From Child 1 ";
29. write(fd[1], msg1, LEN);
30. printf("Child 1: writen [%s] to Pipe\n", msg1);
31. exit(0);
32. }
33. // Child 2
34. **int** child2 = fork();
35. **if** (child2 == -1)
36. {
37. perror("Couldn't fork.");
38. exit(1);
39. }
40. **else** **if** (child2 == 0)
41. {
42. close(fd[0]);
43. **char** msg2[] = "Hello From Child 2 ";
44. write(fd[1], msg2, LEN);
45. printf("Child 2: writen [%s] to Pipe\n", msg2);
46. exit(0);
47. }
48. // Parentss
50. **int** status1, status2;
51. pid\_t ret1 = wait(&status1);
52. pid\_t ret2 = wait(&status2);
54. close(fd[1]);
55. **char** msg1[LEN];
56. read(fd[0], msg1, LEN);
57. **char** msg2[LEN];
58. read(fd[0], msg2, LEN);
59. printf("Parent: read from Pipe [%s%s]\n", msg1, msg2);
60. **if** (WIFEXITED(status1))
61. printf("Parent: child %d finished with %d code.\n", ret1, WEXITSTATUS(status1));
62. **else** **if** (WIFSIGNALED(status1))
63. printf("Parent: child %d finished from signal with %d code.\n", ret1, WTERMSIG(status1));
64. **else** **if** (WIFSTOPPED(status1))
65. printf("Parent: child %d finished from signal with %d code.\n", ret1, WSTOPSIG(status1));
67. **if** (WIFEXITED(status2))
68. printf("Parent: child %d finished with %d code.\n", ret2, WEXITSTATUS(status2));
69. **else** **if** (WIFSIGNALED(status2))
70. printf("Parent: child %d finished from signal with %d code.\n", ret2, WTERMSIG(status2));
71. **else** **if** (WIFSTOPPED(status2))
72. printf("Parent: child %d finished from signal with %d code.\n", ret2, WSTOPSIG(status2));
73. **return** 0;
74. }



**Задание 5:**

В программу с программным каналом включить собственный обработчик сигнала. Использовать сигнал для изменения хода выполнения программы.

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <string.h>
5. #include <signal.h>
6. #include <time.h>
7. #include <sys/wait.h>
9. #define LEN 100
10. #define DELAY 2
12. **int** p\_flag = 0;
14. **void** catch\_sig(**int** signum) {
15. printf("Proccess %d: catched signal %d\n", getpid(), signum);
16. p\_flag = 1;
17. }
19. **int** main() {
20. //Pipe
21. **int** fd[2];
22. **if** (pipe(fd) == -1)
23. {
24. perror("Couldn't pipe.");
25. exit(1);
26. }
27. // Ctrl + Z
28. **void** (\*old\_handler)(**int**) = signal(SIGTSTP, catch\_sig);
30. **pid\_t** child = fork();
31. **if** (child == -1)
32. {
33. perror("Couldn't fork.");
34. exit(1);
35. }
36. **else** **if** (child == 0) // Child
37. {
38. **while** (!p\_flag) ;
39. **char** msg[LEN];
40. close(fd[1]);
41. read(fd[0], msg, LEN);
42. printf("Child: read [%s]\n", msg);
43. sleep(DELAY);
44. }
45. **else** // Parent
46. {
47. printf("Parent: pid = %d\n", getpid());
48. printf("Child: pid = %d\n", child);
49. close(fd[0]);
50. **char** msg[] = "Hello From Parent ";
51. write(fd[1], msg, LEN);
53. printf("Parent: waiting for CTRL+Z signal\n");
54. **while** (!p\_flag);
56. **int** status;
57. pid\_t ret = wait(&status);
59. **if** (WIFEXITED(status))
60. printf("Parent: child %d finished with %d code.\n", ret, WEXITSTATUS(status));
61. **else** **if** (WIFSIGNALED(status))
62. printf("Parent: child %d finished from signal with %d code.\n", ret, WTERMSIG(status));
63. **else** **if** (WIFSTOPPED(status))
64. printf("Parent: child %d finished from signal with %d code.\n", ret, WSTOPSIG(status));
65. }
67. signal(SIGTSTP, old\_handler);
68. **return** 0;
69. }

