HW1

1 #include<stdio.h>

2 #include<unistd.h>

3 #include<stdlib.h>

4   
5

6 int main()

7 {

8 int x = 100;

9 printf("x = %d, (pid:%d)\n", x, (int)getpid());

10 int rc = fork();

11 if (rc < 0) {

12 fprintf(stderr, "fork failed");

13 exit(1);

14 } else if (rc == 0) {

15 printf("child x = %d (pid:%d)\n", x, (int)getpid());

16 x = 200;

17 printf("child after changed x = %d (pid:%d)\n", x, (int)getpid());

18 } else {

19 printf("parent x = %d (pid:%d)\n", x, (int)getpid());

20 x = 300;

21 printf("parent after changed x = %d (pid:%d)\n", x, (int)getpid());

22 }

23 return 0;

24 }

The 8th line declares x=100, the 9th line first prints the identification code of x, uses getpid() to obtain the process identification code of x, the 10th line uses fork() to generate the parent-child program, and the 11th line uses if to judge whether it is Parent or child program, less than means an error occurs, execute exit(1), return the error and end the program, the 14th line is equal to the child, after printing the text we want, jump to line 23 to end the program, if it is not equal, it is not Less than, expressed as the parent, after printing the text we want, jump to line 23 to end the program

HW2

1 #include<stdio.h>

2 #include<unistd.h>

3 #include<stdlib.h>

4 #include<fcntl.h>

5 #include<string.h>

6 #include<sys/wait.h>

7   
8

9 int main()

10 {

11 int fd = open("./5.2.txt", O\_CREAT|O\_WRONLY|O\_TRUNC, S\_IRWXU);

12 int rc = fork();

13 if (rc < 0) {

14 close(fd);

15 fprintf(stderr, "fork failed");

16 exit(1);

17 } else if (rc == 0) {

18 char \* s = "child write something!\n";

19 write(fd, s, strlen(s));

20 } else {

21 char \* s = "parent write something\n";

22 write(fd, s, strlen(s));

23 wait(NULL);

24 close(fd);

25 }

26 return 0;

27 }

Line 11 first opens a write-only txt file. If the file is not there, create one. If the opened file exists and is a general file, it will be truncated to a length of 0 after opening and give him permission. Line 12 uses fork( ) to generate a parent-child file. Line 13 judges whether there is an error in this file. If an error occurs, close the txt file, and use fprintf to indicate that he is wrong, then use exit(1) to close and return the error value, line 17 If ==0 is a sub-file. After writing the text to be written in the file, jump to line 26. If the 20th line is not a sub-file or an error occurs, then this file is the parent file. After writing the text to be written, Close the file, jump to line 26, return 0

HW3

1 #include<stdio.h>

2 #include<stdlib.h>

3 #include<unistd.h>

4

5 int main()

6 {

7 int rc = vfork();

8 if (rc < 0) {

9 fprintf(stderr, "fork failed");

10 exit(1);

11 } else if (rc == 0) {

12 printf("hello\n");

13 exit(1);

14 } else {

15 printf("goodbye\n");

16 }

17 return 0;

18 }

The 7th line uses vfork() to share the file space, the 8th line judges whether an error occurs, if an error occurs, use fprintf() to write the error message, execute the 10th line of exit(1) to close and return an error, the first Line 11 judges whether it is a subprogram, prints hello, closes it with exit(1) and returns an error, line 14, if it is a parent program, prints goodbye, closes the program and returns 0

HW5

1 #include<stdio.h>

2 #include<unistd.h>

3 #include<sys/wait.h>

4 #include<stdlib.h>

5

6 int main()

7 {

8 int rc = fork();

9 int wc = wait(NULL);

10 if(rc < 0) {

11 fprintf(stderr, "fork failed");

12 exit(1);

13 } else if (rc == 0) {

14 printf("child");

15 } else {

16 printf("parent");

17 }

18 printf("pid:%d wc:%d rc:%d\n", (int)getpid(), wc, rc);

19 return 0;

20 }

Line 8 uses fork() to generate parent and child programs, line 9 uses wait(NULL) to wait for all programs to end, and line 10 determines whether an error has occurred. If an error occurs, use fprintf() to generate an error message, and use exit(1 ) If the 13th line == 0, then the subroutine, the printout will jump to the 18th line to print out the pid of the subprogram, the end return message of wc, and the 0 of rc, and if the 15th line is not an error, message, print out the parent and jump to line 18 to print the pid of the parent program, and the end message of wc, and rc's >1, and return 0 on line 20

HW6

1 #include<stdio.h>

2 #include<unistd.h>

3 #include<sys/wait.h>

4 #include<stdlib.h>

5

6 int main()

7 {

8 int rc = fork();

9 int wc = waitpid(rc, NULL, 0);

10 if(rc < 0) {

11 fprintf(stderr, "fork failed");

12 exit(1);

13 } else if (rc == 0) {

14 printf("child");

15 } else {

16 printf("parent");

17 }

18 printf("pid:%d wc:%d rc:%d\n", (int)getpid(), wc, rc);

19 return 0;

20 }

Line 8 uses fork() to generate the parent-child program, line 9 uses waitpid() to monitor the status of the subprogram, line 10 judges whether an error occurs, if an error occurs, use fprintf() to generate an error message, and use exit(1 ) If line 13 == 0, it is a subprogram, print out child and jump to line 18 to print out the pid of the subprogram, as well as the end return message of wc, and 0 of rc, if line 15 is not an error, also message, print out the parent and jump to line 18 to print the pid of the parent program, and the end message of wc, and rc's >1, and return 0 on line 20

HW7

1 #include<stdio.h>

2 #include<unistd.h>

3 #include<stdlib.h>

4

5 int main()

6 {

7 int rc = fork();

8 if (rc < 0) {

9 fprintf(stderr, "fork failed");

10 exit(1);

11 } else if (rc == 0) {

12 close(STDOUT\_FILENO);

13 printf("output child\n");

14 }

15 wait(NULL);

16 return 0;

17 }

Line 7 uses fork() to generate a parent-child program, and line 8 judges whether an error occurs. If an error occurs, use fprintf() to generate an error message, and use exit(1). If line 11 == 0, it is a subroutine, Turn off STDOUT\_FILENO, print output child line 15 uses wait(NULL) to wait for all programs to end, return 0 on line 16