On this page

实验2 创建进程

实验内容与任务

- 使用fork函数创建进程
- 使用wait函数等待子进程结束
- 为父子进程分配计算任务

相关知识及背景

使用fork函数创建子进程时，创建的子进程进程地址空间是父进程地址空间的复本， 即子进程刚创建时地址空间中的内容与父进程进址空间中的内容完全相同。父子进 程此时唯一的差别仅在于fork函数的返回值不同，因此程序代码中可以通过fork函 数的返回值来区分父子进程，并为父子进程分配不同的计算任务。

实验要求及过程

实验要求

1. 理解使用fork创建子进程的过程
2. 理解wait函数的功能和作用
3. 掌握Linux环境下程序的编译与调试

实验过程

示例1. 使用fork函数创建进程

示例文件fork-01.c

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

#define   MAX_COUNT  200
```

```c
#define   BUF_SIZE   100

char MyStr[BUF_SIZE];

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

     sprintf(MyStr,"%s\n","Hello World!");

     MyStr[4]='a';

     fork();

     pid = getpid();

     printf("MyStr:%s\n", MyStr);

     for (i = 1; i <= MAX_COUNT; i++) {
          sprintf(buf, "This line is from pid %d, value = %d\n", pid, i);
          write(1, buf, strlen(buf));
     }

}
```

示例2 分离父子进程工作内容

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

#define   MAX_COUNT  200

void  ChildProcess(void);                /* child process prototype  */
void  ParentProcess(void);               /* parent process prototype */

void  main(void)
{
     pid_t  pid;

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

```
          ParentProcess();
}

void  ChildProcess(void)
{
     int   i;

     for (i = 1; i <= MAX_COUNT; i++)
          printf("   This line is from child, value = %d\n", i);
     printf("   *** Child process is done ***\n");
}

void  ParentProcess(void)
{
     int   i;

     for (i = 1; i <= MAX_COUNT; i++)
          printf("This line is from parent, value = %d\n", i);
     printf("*** Parent is done ***\n");
}
```

示例3 使用wait函数等待子进程

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

#define   MAX_COUNT  200
#define   BUF_SIZE   100

void  ChildProcess(char [], char []);    /* child process prototype  */

void  main(void)
{
     pid_t   pid1, pid2, pid;
     int     status;
     int     i;
     char    buf[BUF_SIZE];

     printf("*** Parent is about to fork process 1 ***\n");
     if ((pid1 = fork()) < 0) {
          printf("Failed to fork process 1\n");
          exit(1);
     }
     else if (pid1 == 0)
```

```
          ChildProcess("First", "   ");

     printf("*** Parent is about to fork process 2 ***\n");
     if ((pid2 = fork()) < 0) {
          printf("Failed to fork process 2\n");
          exit(1);
     }
     else if (pid2 == 0)
          ChildProcess("Second", "      ");

     sprintf(buf, "*** Parent enters waiting status .....\n");
     write(1, buf, strlen(buf));

     pid = wait(&status);
     sprintf(buf, "*** Parent detects process %d was done ***\n", pid);
     write(1, buf, strlen(buf));
     printf("***====> Child Process Status:%d\n", status);

     pid = wait(&status);
     printf("*** Parent detects process %d is done ***\n", pid);
     printf("*** Parent exits ***\n");
     exit(0);
}

void  ChildProcess(char *number, char *space)
{
     pid_t  pid;
     int    i;
     char   buf[BUF_SIZE];

     pid = getpid();
     sprintf(buf, "%s%s child process starts (pid = %d)\n",
             space, number, pid);
     write(1, buf, strlen(buf));
     for (i = 1; i <= MAX_COUNT; i++) {
          sprintf(buf, "%s%s child's output, value = %d\n", space, number, i);
          write(1, buf, strlen(buf));
     }
     sprintf(buf, "%s%s child (pid = %d) is about to exit\n",
             space, number, pid);
     write(1, buf, strlen(buf));
     exit(-1);
}
```

示例4. 父进程也加入了计算内容

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

#define   MAX_COUNT  200
#define   BUF_SIZE   100

void  ChildProcess(char [], char []);    /* 子进程函数  */
void  ParentProcess(void);               /* 父进程函数 */

void  main(void)
{
     pid_t   pid1, pid2, pid;
     int     status;
     int     i;
     char    buf[BUF_SIZE];

     printf("*** Parent is about to fork process 1 ***\n");
     if ((pid1 = fork()) < 0) {
          printf("Failed to fork process 1\n");
          exit(1);
     }
     else if (pid1 == 0)
          ChildProcess("First", "   ");

     printf("*** Parent is about to fork process 2 ***\n");
     if ((pid2 = fork()) < 0) {
          printf("Failed to fork process 2\n");
          exit(1);
     }
     else if (pid2 == 0)
          ChildProcess("Second", "      ");

     ParentProcess();
     sprintf(buf, "*** Parent enters waiting status .....\n");
     write(1, buf, strlen(buf));
     pid = wait(&status);
     sprintf(buf, "*** Parent detects process %d was done ***\n", pid);
     write(1, buf, strlen(buf));
     pid = wait(&status);
     printf("*** Parent detects process %d is done ***\n", pid);
     printf("*** Parent exits ***\n");
     exit(0);
}
```

```c
#define  QUAD(x)  (x*x*x*x)

void  ParentProcess(void)
{
     int  a, b, c, d;
     int  abcd, a4b4c4d4;
     int  count = 0;
     char buf[BUF_SIZE];

     sprintf(buf, "Parent is about to compute the Armstrong numbers\n");
     write(1, buf, strlen(buf));
     for (a = 0; a <= 9; a++)
          for (b = 0; b <= 9; b++)
               for (c = 0; c <= 9; c++)
                    for (d = 0; d <= 9; d++) {
                         abcd     = a*1000 + b*100 + c*10 + d;
                         a4b4c4d4 = QUAD(a) + QUAD(b) + QUAD(c) + QUAD(d);
                         if (abcd == a4b4c4d4) {
                              sprintf(buf, "From parent: "
                                      "the %d Armstrong number is %d\n",
                                      ++count, abcd);
                              write(1, buf, strlen(buf));
                         }
                    }
     sprintf(buf, "From parent: there are %d Armstrong numbers\n", count);
     write(1, buf, strlen(buf));
}

void  ChildProcess(char *number, char *space)
{
     pid_t  pid;
     int    i;
     char   buf[BUF_SIZE];

     pid = getpid();
     sprintf(buf, "%s%s child process starts (pid = %d)\n",
             space, number, pid);
     write(1, buf, strlen(buf));
     for (i = 1; i <= MAX_COUNT; i++) {
          sprintf(buf, "%s%s child's output, value = %d\n",
                  space, number, i);
          write(1, buf, strlen(buf));
     }
     sprintf(buf, "%s%s child (pid = %d) is about to exit\n",
             space, number, pid);
     write(1, buf, strlen(buf));
     exit(0);
}
```