
LUT
University
Operating Systems and System Programming
LAND
OF THE
CURIOUS

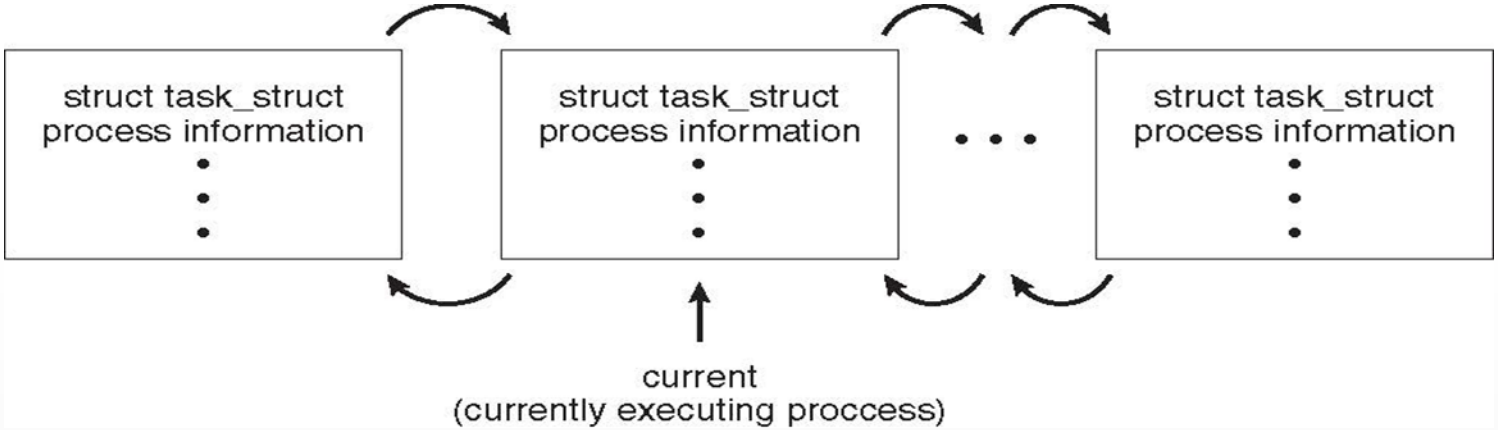
Experiment 4

- Asynchronous concurrent execution of Linux processes/threads
- System calls of creating processes
  - fork()
  - vfork()
  - clone()

Experiment 4

- C task_struct (PCB)
- pid t_pid; /* process identifier */
  long state; /* state of the process */
  unsigned int time_slice /* scheduling information */
  struct task_struct *parent; /* this process's parent */
  struct list_head children; /* this process's children */
  struct files_struct *files; /* list of open files */
  struct mm_struct *mm; /* address space of this process
- ……

Experiment 4



Experiment 4

- fork() creates a process that is almost identical to the original process through a system call.
  - Two processes can do exactly the same thing.
  - Depending on the initial parameters or the variables passed in, the two processes can also do different things.
- System first allocates resources to the new process, such as space for storing data and code. Then copy all the values from the original process to the new process, only a few values are different from the original process.

Experiment 4

- return values
  - =0 child process
  - >0 parent process
  - <0 error
- error reason
  - The current number of processes has reached the upper limit specified by the system. errno = EAGAIN
  - system is out of memory. errno=ENOMEM
- fork.c

Experiment 4

```c
#include <unistd.h>
#include <stdio.h>
int main()
{
        pid_t son_pid,daughter_pid;
        int count = 1;
        son_pid = fork();
        if(son_pid == 0){
                count ++;
                printf("i am son, count=%d\n", count);
        }else{
                daughter_pid = fork();
                if(daughter_pid == 0){
                        count ++;
                        printf("i am daughter, count=%d\n", count);
                }else{
                        count++;
                        printf("i am father, count=%d\n", count);
                        waitpid(son_pid, NULL, 0);
                        waitpid(daughter_pid, NULL, 0);
                }
        }
}
```

- Observe the sequence of the results displayed on the screen until a different sequence appears.

Experiment 4

```
#include <unistd.h>
#include <stdio.h>

int main(){
        pid_t son_pid, daughter_pid;
        int count = 1;
        for (int i=1; i<3; i++){
                son_pid = fork();
                if (son_pid == 0){
                        count++;
                        printf("I am son, count = %d\n", count);
                }
                else{
                        daughter_pid = fork();
                        if (daughter_pid == 0){
                                count++;
                                printf("I am daughter, count = %d\n", count);
                        }
                        else{
                                count++;
                                printf("I am father, count = %d\n", count);
                                waitpid(son_pid, NULL, 0);
                                waitpid(daughter_pid, NULL, 0);
                        }
                }
        }
}
```

Analyze the reasons for the running results.

Experiment 4

```c
#include <unistd.h>
#include <stdio.h>
#include <pthread.h>
void *daughter(void *num)
{
        int* a = (int *)num;
        *a += 1;
        printf("I am daughter,count=%d\n", *a);
}
void *son(void *num)
{
        int* a = (int *)num;
        *a += 1;
        printf("I am son,count=%d\n", *a);
}

int main()
{
        pthread_t son_tid, daughter_tid;
        int count = 1;

        pthread_create(&son_tid, NULL, son, &count);
        pthread_create(&daughter_tid, NULL, daughter, &count);

        count ++;
        printf("I am parent,count:=%d\n", count);
        pthread_join(son_tid, NULL);
        pthread_join(daughter_tid, NULL);

        return 0;
}
```

- Observe the sequence of the results displayed on the screen until a different sequence appears.