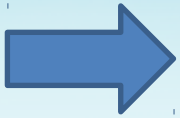
High Performance Distributed Systems Lab

Linux Programming 程序

程序

◆ 每個程序位置以一個特定號碼指定



PID 2~32768

檢視程序

ps 命令可以顯示執行中的程序、另一位使用者正在執行的程序或所有執行中的程序

- –f　：全格式。
- -e　：顯示所有進程。
- a　：顯示終端上的所有進程，包括其他用戶的　　進程。
- x　：顯示沒有控制終端的進程
- TTY　：　顯示啟動處理程序時所在的終端機。
- TIME　：　顯示消耗掉的 CPU 時間。
- CMD　：　顯示啟動處理程序的命令。

STAT ，程序的狀態

- R (Running)：該程式正在運作中；
- S (Sleep)：該程式目前正在睡眠狀態(idle)，但可以被喚醒(signal)。
- D ：不可被喚醒的睡眠狀態，通常這支程式可能在等待 I/O 的情況(ex> 列印)
- T ：停止狀態(stop)，可能是在工作控制(背景暫停)或除錯 (traced) 狀態；

- #include <stdlib.h>
- int system (const char *string);

```c
#include <stdlib.h>
#include <stdio.h>

int main()
{
     printf("Running ps with system\n");
     system("ps ax");
     printf("Done.\n");
     exit(0);
}
~
```

```
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:00 /sbin/init
    2 ?        S      0:00 [kthreadd]
    3 ?        S      0:00 [migration/0]
    4 ?        S      0:00 [ksoftirqd/0]
    5 ?        S      0:00 [watchdog/0]
    6 ?        S      0:00 [migration/1]
    7 ?        S      0:00 [ksoftirqd/1]
    8 ?        S      0:00 [watchdog/1]
    9 ?        S      0:00 [migration/2]
   10 ?        S      0:00 [ksoftirqd/2]
   11 ?        S      0:00 [watchdog/2]
   12 ?        S      0:00 [migration/3]
   13 ?        S      0:00 [ksoftirqd/3]
   14 ?        S      0:00 [watchdog/3]
   15 ?        S      0:09 [events/0]
   16 ?        S      0:43 [events/1]
   17 ?        S      0:03 [events/2]
   18 ?        S      0:04 [events/3]
   19 ?        S      0:00 [cpuset]
```

```
29835 ?        Ssl    2:18 /usr/sbin/mysqld
31102 pts/1    T      0:04 vim York_2048_2.c
31552 ?        Ss     0:00 sshd: root@pts/2
31626 pts/2    Ss     0:00 -bash
31815 pts/1    S+     0:29 vim -r York_2048_2.c
Done.
```

程序的替換

- 有一系列的相關函數，都是以 exec 為首。他們起動程式的方法不同還包含程式的參數。 exec 函數會以一個新的處理程序取代目前的處理程序，新處理程序就是 path 或 file 參數所指定程式。

- #include<unistd>
- int execl(const char *path, const char *arg0, ..., (char *) 0);
- int execlp(const char *file, const char *arg0, ..., (char *) 0);
- int execle(const char *path, const char *arg0, ..., (char *) 0, const char *envp[]);
- int execv(const char *path, const char *argv[]);
- int execvp(const char *file, const char *argv[]);
- int execve(const char *path, const char *argv[], const char *envp[]);

- path 參數表示你要啟動程序的名稱。
- arg 參數表示啟動程序所帶的參數，一般第一個參數為要執行命令名，不是帶路徑且 arg 必須以 NULL 結束。
- 帶 l 的 exec 函數： execl, execlp, execle ，表示後邊的參數以可變參數的形式給出且都以一個空指針結束。
- 字尾為 p 的函數期不同之處在於他們會搜尋 PATH 環境變數來尋找新程式的執行檔。

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

int main(void)
{
   printf("entering main process---\n");
   execl("/bin/ls","ls","-l",NULL);
   printf("exiting main process ----\n");
   return 0;
}
```

```
[zxy@test unixenv_c]$ cc execl.c
[zxy@test unixenv_c]$ ./a.out
entering main process---
total 104
-rwxrwxr-x. 1 zxy zxy  5976 Jul 12 22:54 a.out
-rw-r--r--. 1 zxy zxy   527 Jul 12 15:48 atexit02.c
-rw-r--r--. 1 zxy zxy   426 Jul 12 15:59 atexit03.c
-rw-r--r--. 1 zxy zxy   287 Jul 12 15:44 atexit.c
-rw-r--r--. 1 zxy zxy   472 Jul 10 12:39 creathole.c
```

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

int main(void)
{
    printf("entering main process---\n");
    int ret;
    char *argv[] = {"ls","-l",NULL};
    ret = execvp("ls",argv);
    if(ret == -1)
        perror("execl error");
    printf("exiting main process ----\n");
    return 0;
}
```

```
[zxy@test unixenv_c]$ cc execl.c
[zxy@test unixenv_c]$ ./a.out
entering main process---
total 104
-rwxrwxr-x. 1 zxy zxy       5976 Jul 12 22:54 a.out
-rw-r--r--. 1 zxy zxy        527 Jul 12 15:48 atexit02.c
-rw-r--r--. 1 zxy zxy        426 Jul 12 15:59 atexit03.c
-rw-r--r--. 1 zxy zxy        287 Jul 12 15:44 atexit.c
-rw-r--r--. 1 zxy zxy        472 Jul 10 12:39 creathole.c
```

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

int main(int argc, char *argv[])
{
   //char * const envp[] = {"AA=11", "BB=22",
NULL};
   printf("Entering main ...\n");
   int ret;
   ret =execl("./hello", "hello", NULL);
   //execle("./hello", "hello", NULL, envp);
   if(ret == -1)
      perror("execl error");
   printf("Exiting main ...\n");
   return 0;
```

```
[zxy@test unixenv_c]$ cc execle.c -o execle
[zxy@test unixenv_c]$ cc hello.c -o hello
[zxy@test unixenv_c]$ ./execle
Entering main ...
hello pid=4267
HOSTNAME=test
SELINUX_ROLE_REQUESTED=
TERM=xterm
SHELL=/bin/bash
HISTSIZE=1000
SSH_CLIENT=172.17.1.112 61324 22
SELINUX_USE_CURRENT_RANGE=
QTDIR=/usr/lib/qt-3.3
QTINC=/usr/lib/qt-3.3/include
SSH_TTY=/dev/pts/0
```

```c
#include <unistd.h>
#include <stdio.h>
extern char** environ;

int main(void)
{
   printf("hello pid=%d\n", getpid());
   int i;
   for (i=0; environ[i]!=NULL; ++i)
   {
      printf("%s\n", environ[i]);
   }
return 0;
```

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

int main()
{
    printf("Running ps with execlp\n");
    execlp("ps", "ps", "ax", 0);
    printf("Done.\n");
    exit(0);
}
~
~
```

- 程式在顯示第一個訊息後，呼叫 execlp，搜尋由 ps 程式的 PATH 環境變數所指定的路徑，隨後執行 ps 程式，取代原本的程式。
- 沒有 Done 的資訊出現。

複製程序

- 呼叫 fork 來建立新的程序，這個系統呼叫會複製目前的程序，在程序表中建立以目前執行程序有相同屬性的新程序。

- #include<sys/types.h>
- #include<unisted.h>
- pid_t fork(void);

```
pid_t  new_pid ;
new_pid = fork( );
Switch(new_pid) {
case -1 :   //error
        break;
case  0  :  //we are child
        break;
default     :  //we are parent
        break;
}
```

```c
int main()
{
    pid_t pid;
    char *message;
    int n;

    printf("fork program starting\n");
    pid = fork();
    switch(pid)
    {
    case -1:
        perror("fork failed");
        exit(1);
    case 0:
        message = "This is the child";
        n = 5;
        break;
    default:
        message = "This is the parent";
        n = 3;
        break;
    }

    for(; n > 0; n--) {
        puts(message);
        sleep(1);
    }
    exit(0);
```

呼叫 fork 時，這個程式會變成兩個獨立的處理程序父處理程序的辨識方式是經由 fork 回傳的的非零值

```
root@cuda04:~/Frank/ex/ch10# ./fork1
fork program starting
This is the parent
This is the child
This is the parent
This is the child
This is the parent
This is the child
root@cuda04:~/Frank/ex/ch10# This is the child
This is the child
```

等待程序

- #include<sys/types.h>
- #include<sys/wait.h>
- pid_t wait(int *stat_loc);
- wait 系統呼叫會讓父處理程序暫停，直到他的子處理程序結束。這個函數會回傳子處理程序的 PID 。狀態資訊，讓父處理程序可以判斷子處理程序的結束狀態，也就是 exit 回傳值。
- loc 的值並不是 null 指標，所以狀態資訊會被寫入目前所指的位置。

- WIFEXITED(stat_val)　：子程序正常終止傳回非零值。
- WEXITSTATUS(stat_val): WIFSIGNALED 非零，傳回子程序離開碼。
- WIFSIGNALED(stat_val): 子程序因漏失訊息而終止則為非零。
- WTERMSIG(stat_val)　: WIFSIGNALED 非零則傳回訊息號碼。
- WIFSTOPPED(stat_val) : 子程序停止則為非零。
- WSTOPSIG(stat_val)　: WIFSTOPPED 非零則傳回訊　息號碼。

```c
int main()
{
    pid_t pid;
    char *message;
    int n;
    int exit_code;

    printf("fork program starting\n");
    pid = fork();
    switch(pid)
    {
    case -1:
        exit(1);
    case 0:
        message = "This is the child";
        n = 5;
        exit_code = 37;
        break;
    default:
        message = "This is the parent";
        n = 3;
        exit_code = 0;
        break;
    }

    for(; n > 0; n--) {
        puts(message);
        sleep(1);
    }

    This section of the program waits for the child process to finish.  */

    if(pid!=0) {
        int stat_val;
        pid_t child_pid;

        child_pid = wait(&stat_val);

        printf("Child has finished: PID = %d\n", child_pid);
        if(WIFEXITED(stat_val))
            printf("Child exited with code %d\n", WEXITSTATUS(stat_val));
        else
            printf("Child terminated abnormally\n");
    }
    exit (exit_code);
}
```

```
root@cuda04:~/Frank/ex/ch10# ./wait
fork program starting
This is the parent
This is the child
This is the parent
This is the child
This is the parent
This is the child
This is the child
This is the child
Child has finished: PID = 3744
Child exited with code 37
```

- 父處理程序從 fork 取得一個非零值，隨後使用 wait 暫停本身的程式，等待子處理程序回復狀態資訊。

輸出入的轉向

```c
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>

int main()
{
    int ch;
    while((ch = getchar()) != EOF) {
        putchar(toupper(ch));
    }
    exit(0);
}
~
~
```

```
root@cuda04:~/Frank/ex/ch10# ./upper
hello
HELLO
```

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

int main(int argc, char *argv[])
{
    char *filename;

    if(argc != 2) {
        fprintf(stderr, "usage: useupper file\n");
        exit(1);
    }

    filename = argv[1];

    if(!freopen(filename, "r", stdin)) {
        fprintf(stderr, "could not redirect stdin to file %s\n", filename);
        exit(2);
    }

    execl("./upper", "upper", 0);

    perror("could not exec ./upper");
    exit(3);
}
~
```

```
root@cuda04:~/Frank/ex/ch10# ./useupper file.txt
THIS IS THE FILE, FILE.TXT; IT IS ALL LOWER CASE.
```

- useupper 使用 freopen 來關閉標準輸入，並使用程式引數來關聯檔案資料流 stdin ，呼叫 execl 以上面的程式來取代執行中的程序。

訊息名稱，signal.h

- SIGABORT * 程序停止
- SIGALRM 警示
- SIGFPE * 浮點數例外
- SIGHUP 掛斷
- SIGILL * 非法指令
- SIGINT 終端機插斷

Ctrl-C (in older Unixes, DEL) sends an INT signal (SIGINT); by default, t his causes the process to terminate.
Ctrl-Z sends a TSTP signal (SIGTSTP); by default, this causes the proces s to suspend execution.
Ctrl-\ sends a QUIT signal (SIGQUIT); by default, this causes the proces s to terminate and dump core.

- #include<signal.h>
- void(*signal(int sig,void (*func)(int)))(int) ;
- 兩個參數 : sig 與 func，sig指名要處理的訊號類型，func描述與信號關聯的動作。
- SIG_IGN: 表示忽略該信號，執行signal()後，進　程會忽略類型為sig的信號。
- SIG_DFL: 這個符號表示恢復系統對信號的默認處　　理。

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

void ouch(int sig)
{
    printf("OUCH! - I got signal %d\n", sig);
    (void) signal(SIGINT, SIG_DFL);
}

int main()
{
    (void) signal(SIGINT, ouch);

    while(1) {
        printf("Hello World!\n");
        sleep(1);
    }
}
~
```

```
root@cuda04:~/Frank/ex/ch10# ./ctrlc1
Hello World!
Hello World!
Hello World!
^COUCH! - I got signal 2
Hello World!
Hello World!
Hello World!
^C
```

- int sigaction(int sig,const struct sigaction *act, struct sigaction *oact);
  sig: 要處理的訊號，若 act 指針非空， 則根據 act 修改該信號的處理動作。若 oact 指針非空，則通過 oact 傳出該信號原來的處理動作。
- Void(*)(int) sa_handler
- sigset_t sa_mask
- int sa_flags
- sa_handler 代表新的信號處理。
- sa_mask 用來設置在處理該信號時暫時將 sa_mask 指定的信號集擱置。
- sa_flags 用來設置信號處理的其他相關操作。

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

void ouch(int sig)
{
    printf("OUCH! - I got signal %d\n", sig);
}

int main()
{
    struct sigaction act;

    act.sa_handler = ouch;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;

    sigaction(SIGINT, &act, 0);

  while(1) {
    printf("Hello World!\n");
    sleep(1);
  }
}
```

```
root@cuda04:~/Frank/ex/ch10# ./ctrlc2
Hello World!
Hello World!
Hello World!
Hello World!
^COUCH! - I got signal 2
Hello World!
Hello World!
Hello World!
^COUCH! - I got signal 2
Hello World!
Hello World!
Hello World!
Hello World!
^\離開
```

Thanks for your listening ! !