Code Example for Shell Lab

Lei Tian

Solution for some errors

- Errors like “undefined reference to ‘pthread_create’”
  - Add “LDLIBS = -lpthread” in Makefile
- Errors like “‘for’ loop initial declarations are only allowed …”
  - for (int i=0; i<5; i++) ->   int i=0; for(i=0; i<5; i++)
- Errors like “A NULL argv[0] was passed through …. ”
  - original: Execve(“/bin/date”, NULL, NULL);
  - solution : char *args [] = { “/bin/date”, NULL, NULL, NULL};

    Execve(args[0], args, NULL);

Children-do-not-wait

- ./children-do-not-wait [n]
  - fork n child processes
- Parent: print n times
  - [parentpid]: fork child
  - [parentpid]: Child is [childpid]
- Child
  - [childpid]: I am the child

Children-do-not-wait

- Order is not guaranteed between children
  - “[childpid_m]: I am the child”
  - “[parentpid]: Child is [childpid_n]” (childpid_m < childpid_n)
- Order is even not guaranteed within one child !
  - “[parentpid]: Child is [childpid]” (mostly first)
  - “[childpid]: I am the child”
- Shell unblocks once parent process finishes
  - “[childpid]: I am the child” could comes after shell unblockes

Children-simple-wait

- waitpid(-1, null , 0)
    - wait for any child to finish then unblock
    - if any, return the positive pid
    - if not, return -1

Children-simple-wait

- ./children-simple-wait [n]
  - forks n child processes
- Parent: print n times
  - [parentpid]: fork child
  - [parentpid]: Child is [childpid]
  - [parentpid]: reap child [childpid]
- Child
  - [childpid]: I am the child

Children-simple-wait

- Order is guaranteed within one child for
  - “[childpid]: I am the child” (always first)
  - “[parentpid]: reap child [childpid]”
- Order is not guaranteed between children
  - “[childpid_m]: I am the child”
  - “[parentpid]: reap child [childpid_n]” (childpid_m > childpid_n)

Children-sigchild-no-wait

- Signal(SIGCHLD, handler) (Recommended)
  - parent registers *handler* func to response once receives SIGCHLD signal - **non-blocking** way
  - child sends SIGCHLD signal to parent once it finishes execution, ex: exit(0)
  - once registered, responses anytime if new signal arrives
  - *handler* func still runs in parent process
  - unregister once parent finishes, ex: exit(0)

Children-sigchild-no-wait

- pause()
  - suspends program execution until any signal comes
  - actions for signal:
    1. executes a handler function
    2. terminates the process
  - explicitly waits for children termination - **blocking** way
  - cooperates with *Signal* func to reap children

Children-sigchild-no-wait

- ./children-sigchild-no-wait [n] [m]
  - forks n child processes
  - m!=0: parent call *pause()* wait for signal from children
  - By default, m==0, not wait for signal from children
- When m==0, could miss reaping children
  - parent finishes before some child processes
  - could missing all when n is small !!!

Children-sigchild-correct-wait

- ./children-sigchild-correct-wait [n]
  - forks n child processes
  - global param *died* keeps track of reaped children
  - *died* need update in handler func
  - parent waits until *died* reaches # of created children

Children-sigchild-correct-wait

```
void handler (int sig){
        pid_t pid;
        while((pid==waitpid(-1, ...))>0) {
           … …
           died ++;
        }
}
//in main
while(died < kids){
        pause();
}
```

- **Always** Use **while** instead of **if** !!!
  - anytime only one handler func is executed
  - when handler executes inside *while* loop, it is possible new child finishes and new signal arrives, using *if* will miss this signal, cause *died* unsynchronized and finally block parent process in *pause()*

Procmask-before

- ./procmask-before [n]
  - By default (n==0)
    - no time waiting for signal from children, parent finishes before any child is execute, no “deletejob” message
    - output by children is displayed after shell unblocks
  - When n>0,
    - parent waits *n* seconds for SIGCHLD from each child and then “deletejob”
    - “deletejob” comes before “addjob”

Procmask-partial

```
Signal(SIGCHLD, handler);
   … …
Sigemptyset(&mask);
Sigaddset(&mask, SIGCHLD);
Sigprocmask(SIG_BLOCK, &mask,
NULL);
Sigprocmask(SIG_UNBLOCK, &mask,
NULL);
```

1. register *handler* to response SIGCHLD signal
2. initialize and empty an signal set (mask)
3. add SIGCHLD to signal set
4. block all signals in signal set
   - SIGCHLD is blocked and not delivered to parent until unblock
5. unblock all signal in signal set
   - once unblocked, SIGCHLD immediately reaches parent, triggers *handler* to response

Procmask-partial

- ./procmask-partial [n]
  - Guarantees “deletejob” after “addjob” by
    - blocks SIGCHLD before fork
    - unblocks SIGCHLD after *sleep* and *addjob*
  - Still by default (n==0), parent finishes before any child, no “deletejob” message

Procmask-show-flaw

```
if((pid==Fork())==0){
        Evecve(“./children-sigchild-correct-wait”, NULL, NULL );
}
```

- Creates a child process and calls *children-sigchild-correct-wait* program inside
- *children-sigchild-correct-wait:*
  - By default, creates one sub-child and reaps it by registering SIGCHILD handler

Procmask-show-flaw

```
if((pid==Fork())==0){
        Evecve(“./children-sigchild-correct-wait”, NULL, NULL );
}
```

- Fork:
  - copies everything from parent to child process
- *children-sigchild-correct-wait* also copies blocking SIGCHLD status
  - prevent calling *handler* forever, no reap!!!

Procmask-show-fix

```
if((pid==Fork())==0){
        Sigprocmask(SIG_UNBLOCK, &mask, NULL);
        Evecve(“./children-sigchild-correct-wait”, NULL, NULL );
}
```

- Unblcok SIGCHLD in *children-sigchild-correct-wait* to trigger *handler* and reap child process

Solution for some errors

- Errors like “undefined reference to ‘pthread_create’”
  - Add “LDLIBS = -lpthread” in Makefile
- Errors like “‘for’ loop initial declarations are only allowed …”
  - for (int i=0; i<5; i++) ->   int i=0; for(i=0; i<5; i++)
- Errors like “A NULL argv[0] was passed through …. ”
  - original: Execve(“/bin/date”, NULL, NULL);
  - solution : char *args [] = { “/bin/date”, NULL, NULL, NULL};

    Execve(args[0], args, NULL);