Process Synchronization: Pipes and Select

- Learning Objectives:
  - Use pipes to synchronize between two processes
  - Identify race conditions when we need to both wait and test a condition.
  - Use select to solve the “wait and check” problem.

Coordinating between processes

- We've seen a number of mechanisms that processes can use to communicate:
  - Signals
  - Process exit (when it's a child)
  - Pipes
- Let's first see how we can use pipes to synchronize two processes.
- Then we're review the problems we encountered in Thursday's class trying to both wait for children to exit and time out the parent if the children take too long.

Terminal

File Edit View Terminal Tabs Help

```
pingpong.c
// This is a the first in a sequence of simple program demonstrating
// the need for synchronization between two processes.  The later version
// solve the problem using pipes.
//
// The parent forks two children, one of whom is supposed to
// print ping to stdout; the other of whom is supposed to print pong. But
// the children must synchronize so that the pings and pongs alternate.
// We construct two pipes -- the ping child must read a token from the ping
// pipe before printing its character; the pong child must read a token from
// the pong pipe before printing its character.









// Unsynchronized version -- pings and pongs will not alternate nicely

main(      )
{
    pid_t p, pingpid, pongpid;
          reaped_ping, reaped_pong, status;
<videos/select/pingpong.c   CWD: /home/ubuntu/cs61/cs61-videos/select   Line: 1
"pingpong.c" 71L, 2043C
```

Terminal

File Edit View Terminal Tabs Help

```
pingpong2.c
{
     token;
     p, pingpipe[ ], pongpipe[ ];
     reaped_ping, reaped_pong;
    pid_t pingpid, pongpid, status;

    reaped_ping = reaped_pong =  ;

    // Create the two pipes
       (pipe(pingpipe) || pipe(pongpipe)) {
        fprintf(      ,                                     ,
                strerror(errno));
    }

    // Set up ping process
     pingpid = fork();
            (pingpid) {
                :
             // Child ping process -- only writes "ping" after a successful
             // read on the ping pipe
             close(pingpipe[ ]);
             close(pongpipe[ ]);
                   ( ) {
                 read(pingpipe[ ], &token,  );
<deos/select/pingpong2.c   CWD: /home/ubuntu/cs61/cs61-videos/select   Line: 39
```

Terminal

File Edit View Terminal Tabs Help

pingpong4.c

```
    // Get things started by writing into the pingpipe and then
    // close the pipes, so there aren't any extraneous opens
    write(pingpipe[ ], &token,  );
    close(pingpipe[ ]);
    close(pingpipe[ ]);
    close(pongpipe[ ]);
    close(pongpipe[ ]);

    sleep( );
    // Check if children are still running; kill if they are.

       (waitpid(pingpid, &status, WNOHANG) ==  )
        kill(pingpid,       );

       (waitpid(pongpid, &status, WNOHANG) ==  )
        kill(pongpid,       );

    printf(                  );
}
```

<deos/select/pingpong4.c   CWD: /home/ubuntu/cs61/cs61-videos/select   Line: 90

```
Terminal
File  Edit  View  Terminal  Tabs  Help
pingpong6.c

    // Set up ping process
     pingpid = fork();
            (pingpid) {
               :
             // Child ping process -- only writes "ping" after a successful
             // read on the ping pipe
             close(pingpipe[ ]);
             close(pongpipe[ ]);
             // Let's set a timer to go off in 1 second and then exit!
            signal(       , alrm_die_handler);
            timerclear(&itimer.it_interval);
            itimer.it_value.tv_sec =  ;
            itimer.it_value.tv_usec =  ;
            ret = setitimer(ITIMER_REAL, &itimer,     );
                   ( ) {
                    (read(pingpipe[ ], &token,  ) !=  )
                          ;
                 printf(        );
                 fflush(      );
                    (write(pongpipe[ ], &token,  ) !=  )
                          ;
             }
                 ;
<deos/select/pingpong6.c   CWD: /home/ubuntu/cs61/cs61-videos/select   Line: 66
```

What’s a Programmer to do???

- This is a frequently occurring problem:
  - We want to check on an event and if the event hasn’t happened yet,
  - We want to sleep
- However, the primitives that we’ve been using: signals and sleeping and waiting are not with respect to the condition that we want to test.
  - In other words, between the time you check for something and the time you sleep, the event can happen,
  - And when that does, you can lose track of the event!

Meet `select`

```
int select(int nfds,
       fd_set *restrict readfds,
       fd_set *restrict writefds,
       fd_set *restrict errorfds,
       struct timeval *restrict timeout);
```

- From the manual page:
  - Select() examines the I/O descriptor sets whose addresses are passed in readfds, writefds, and errorfds to see if some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively. The first nfds descriptors are checked in each set; i.e., the descriptors from 0 through nfds-1 in the descriptor sets are examined. On return, select() replaces the given descriptor sets with subsets consisting of those descriptors that are ready for the requested operation. Select() returns the total number of ready descriptors in all the sets.
  - **If timeout is a non-null pointer, it specifies a maximum interval to wait for the selection to complete.**

How do we use select to solve our problem?

- Our parent isn't waiting on a file descriptor, so how can we use that to solve our problem?
- Solution: let's create a descriptor that we can use!

- The idea:
  - Parent sets up a pipe.
  - Parent sets up a signal handler for SIGCHLD.
  - Parent does a select on the read end of the pipe.
  - In the SIGCHLD handler, write a byte to the pipe!
  - Now – the parent can both wait on the fd used in the handler AND it can set a timeout on it!

File Edit View Terminal Tabs Help

pingpong7.c

```
// timestamp()
//    Return the current time as a double.

               timestamp(   ) {
        timeval tv;
    gettimeofday(&tv,     );
           tv.tv_sec + tv.tv_usec /          ;
}

// Global signal pipe used to communicate between the mainline program and
// the signal handler.
    signalpipe[ ];

// SIGCHLD handler that writes into our pipe
     child_handler(    signal) {
        c;
    (    )signal;
    assert(write(signalpipe[ ], &c,   ) ==   );
}

     alrm_die_handler(    signal) {
    exit( );
}
```

<deos/select/pingpong7.c   CWD: /home/ubuntu/cs61/cs61-videos/select   Line: 44

Wrapping Up

- The `select` system call lets you solve the "check and wait" problem that crops up in many concurrent applications.
- There is a newer version of `select`, called `pselect`. We'll ask you some questions about `pselect` on the pre-class work – read the man page!
- Synchronization is such a common problem that we have a collection of synchronization primitives.
- The common solution to the "check and wait" problem is called a condition variable.