OPERATING SYSTEM TUTORIAL 2

University of Victoria

PART-1

- start looping
- print: “username@hostname: /home/user >”
  - #include <unistd.h>
  - char *getlogin(void);
  - int gethostname(char *name, size_t namelen);
  - char *getcwd(char *buffer, size_t size);

Always check for errors, and free any memory that is allocated
Check manual for each function

PART-1

- read a line from terminal
- if strcmp(line,“exit\n”)==0 (or strcmp(line,“exit”)==10), exit
- execute the input line by:
  - fork
  - execvp

execvp(char* file, char* argv[])

- #include <unistd.h>
- int execvp(const char *file, char *const argv[])
- execvp() provides an array of pointers to null-terminated strings that represent the argument list available to the new program.
- The array of pointers must be terminated by a NULL pointer. (tokenize input line)
- The exec family of functions replaces the current process image with a new process image.
- If execvp() returns -1, an error will have occurred.

University of Victoria

Tokenize strings

- For calling execvp(char* file, char* argv[]), we need to separate input line by either empty space or new line (‘\n’)
- example:

```
argv[0]=strtok(line," \n”);//“ \n” includes space & new line
int i=0;
while(argv[i]!=NULL){
    argv[i+1]=strtok(NULL,” \n");
    i++;
}
```

fork

- why need fork?
- example:

```
pid_t p=fork();
if(p==0) {
    execvp(argv[0], argv);
    printf("ERROR\n");
}
```

Negative Value: The creation of a child process was unsuccessful.
Zero: Returned to the newly created child process.
Positive value: Returned to parent or caller. The value contains the process ID of the newly created child process.

University of Victoria

- parent needs to wait
- if a wait is not performed, resources associated with the child are not released. Then the terminated child remains in a “zombie" state

Wait

- pid_t wait(int *status);
  pid_t waitpid(pid_t pid, int *status, int options);
- pid:
  - <-1: wait for any child process whose group id is |pid|
  - -1: wait for any child process
  - 0: wait for any child process whose group id is equal to that of the calling process
  - >0: wait for the child process whose id is pid
- options:
  - WUNTRACED | WCONTINUED (or 0): return if a child has stopped
  - WNOHANG (or 1): return immediately if no child has exited

Wait

- return:
  - wait: on success, returns the process ID of the terminated child, return -1 if error
  - waitpid: on success, returns the process ID of the child whose state has changed. If WNOHANG was specified and child(ren) have not yet changed state, return 0. Return -1 if error.
- for example:
  - wait for any child until child terminates: wait(NULL) is shorthand for waitpid(-1, NULL, 0)
  - return immediately: waitpid(0, NULL, WNOHANG)
  - wait for certain child: waitpid(child_pid, NULL, 0)

University of Victoria

PART-2

- if strcmp(argv[0],“cd”)==0 PART-2
  else PART-1
- if argv[1]==NULL || strcmp(argv[1],”~”)==0
  go to HOME (env)
  else go to argv[1]
- “echo $HOME” in BASH to verify

PART-3

- strcmp(argv[0],”bg")==0
- new argv: left shift the old argv, end with ‘\0’
- fork:
  - child: execvp(newArgv[0], newArgv)
  - parent:

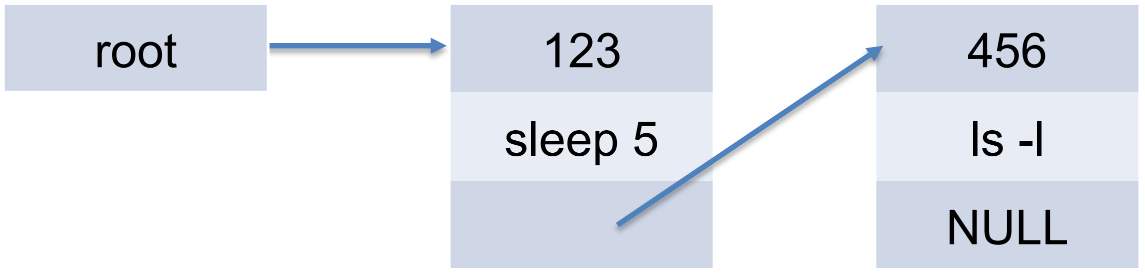
```
if # of bg_process == 0
     root=bg_pro1
else
     append bg_proNext
```

linked list

- linked list for storing bg process info

```
struct bg_pro{
    pid_t pid;
    char command[1024];
    struct bg_pro* next;
};
```

- add, delete



bglist

- strcmp(args[0],"bglist")==0
- print info in the linked list and number of bg processes

check if child terminates

- if # of bg_process > 0
  pid_t ter=waitpid(0,NULL,WNOHANG);
  - if ter > 0 //any child terminates
    - if root -> pid == ter
      - print root -> pid & root -> command & “has terminated”
      - root = root -> next
    - else
      - loop until we find cur -> pid == ter
      - print cur -> pid & cur -> command & “has terminated”
      - cur -> next = cur -> next -> next

Don’t forget to free the removed nodes

Q&A

University of Victoria