SystemPrograming

Homework 2 (**smsh.c**)

컴퓨터공학과
2012036901
윤진한
2016.12.02

1. Source Code

shell.c — UNREGISTERED

```c
/*
    2012036901 윤 진 한
    컴퓨터공학과
    시스템프로그래밍 - 이론 과제 HW2
    < Simple Linux Shell - smsh.c >
    2016.12.01
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <ctype.h>
#include <errno.h>

#define INPUT       STDIN_FILENO
#define OUTPUT      STDOUT_FILENO

#define HISTORYSIZE 10
#define INPUTSIZE   1024
#define INPUTSIZE2  256
#define ARGSIZE     32
#define COMMANDCNT  64
#define PATHSIZE    256
#define TRUE        1
#define FALSE       0

typedef unsigned char bool;

struct sigaction new;                       // background process signal handler 함수.

char commandLine[INPUTSIZE];                // read 로 받은 명령어.
char commandSemi[COMMANDCNT][INPUTSIZE];    // commandLine 을 ";" 으로 파싱.
char commandBg[COMMANDCNT][INPUTSIZE2];     // commandLine 을 각각 Background 로 파싱.
```

Line 235, Column 20 — Tab Size: 4 — C

shell.c — UNREGISTERED

```c
char commandLineArg[COMMANDCNT][ARGSIZE];   // commandLine 을 각각 Argument 로 파싱.
char history[HISTORYSIZE][INPUTSIZE];       // 이전 검색 내용 저장.
char currentPath[PATHSIZE];                 // 현재 경로저장.
char tmpBuffer[INPUTSIZE];                  // 파싱 할때 쓰이는 공동변수.
int backgroundPid[INPUTSIZE2];              // background process pid 저장.
int backgroundCnt;                          // background process count.
bool background;                            // background 실행 여부.

void mainLoopStart(void);                   // shell loop 함수.
bool parseCommandSemicolon(int *);          // ";" 파싱 함수.
void parseCommandArg(char *,int *);         // command 옵션 단위 파싱 함수.
void parseBackground(char *,int *,bool *);  // background 파싱 함수.
void printCurrentPath(void);                // 현재 경로 출력 함수.
void insertHistory(int *);                  // history 에 현재 commandLine 삽입 함수.
void showHistory(int *);                    // history 출력 함수.
void processCommand(char *);                // command 를 실행시켜주는 함수.
void changeDirectory(int);                  // Directory 를 변경시켜주는 함수.
void inoutFileRedirection(int);             // ">"/"<"/">>"/">!" redirection 처리 함수.
void pipeFunction(void);                    // pipe 실행 함수.
void multipipeFunction(int);                // multipipe 실행 함수.
void backgroundDone(void);                  // background process 종료.
void handler(int);                          // background process 종료하기 위한 시그널 핸들러.
void error(char *s);                        // error 발생시 실행 함수.
char *lightTrim(char *);                    // 문자열 좌측 공백 제거 함수.
char *rightTrim(char *);                    // 문자열 우측 공백 제거 함수.
char *trim (char *);                        // 문자열 좌우 공백 모두 제거 함수.
void deleteSpaceOrTab(char *,char);         // 문장에서 Space 나 Tab 을 제거하는 함수.

int main(void){

    new.sa_handler = handler;
    new.sa_flags = SA_NOCLDWAIT;

    sigemptyset(&new.sa_mask);

    sigaction(SIGCHLD,&new,NULL);
    // backgroud child process 가 SIGCHLD 를 보냈을때 new sigaction 동작.
    mainLoopStart();
    // mainLoopStart 함수 실행.
```

Line 40, Column 79 — Tab Size: 4 — C

shell.c — UNREGISTERED

shell.c

```
79      return 0;
80  }
81
82  void mainLoopStart(void){
83
84      int n,historyCnt = -1;
85      int commandLineSemiCnt,cnt,cnt2;
86      bool isLastBackground = FALSE;
87
88      /*----- start of while shell loop. -----*/
89      while (printCurrentPath(),read(INPUT, commandLine, INPUTSIZE)) {
90
91          if ((n==1 && commandLine[0] == '\n') || (n==2 && commandLine[0] == '\t')) continue;
92          // 입력 받은 문자가 엔터나 tab 인 경우.
93          commandLine[strlen(commandLine) - 1] = '\0';
94          // read 로 입력 받을시 마지막에 '\n' 이 입력되어 null 값으로 바꾸어줌.
95          commandLineSemiCnt = 0,cnt = 0;
96          insertHistory(&historyCnt);
97          // 입력받은 명령어는 Queue 형태의 history 에 저장해줌.
98          char *buffer = (char *)malloc(sizeof(char)*INPUTSIZE);
99          bzero(buffer,INPUTSIZE);
100         strcpy(buffer,commandLine);
101         deleteSpaceOrTab(buffer,' ');
102         if (!strlen(buffer)) {
103             free(buffer);
104             continue;
105             // 입력받은 문자에 스페이스만 있는 경우.
106         } else {
107             free(buffer);
108         }
109         if (!parseCommandSemicolon(&commandLineSemiCnt)){
110             bzero(commandLine,INPUTSIZE);
111             bzero(commandSemi,COMMANDCNT*INPUTSIZE);
112             continue;
113         } // 입력받은 명령어를 ';' 으로 파싱함.
114         if (!commandLineSemiCnt){
115             commandLineSemiCnt++;
116             strcpy(commandSemi[0],commandLine);
117         }
```

Line 79, Column 14 — Tab Size: 4 — C

shell.c — UNREGISTERED

shell.c

```
118         while (commandLineSemiCnt--){
119             if (!strcmp(commandSemi[cnt],"exit")) exit(0);
120             if (!strcmp(commandSemi[cnt],"clear")) printf("\033[2J\033[1H");
121             if (!strcmp(commandSemi[cnt],"history")) showHistory(&historyCnt);
122             else {
123                 cnt2 = 0;
124                 background = FALSE;
125                 isLastBackground = FALSE;
126                 bzero(commandBg,COMMANDCNT*INPUTSIZE2);
127                 parseBackground(commandSemi[cnt],&cnt2,&isLastBackground);
128                 // 각각 '&' 실행인지 확인하고 명령어를 실행할 준비를 한다.
129                 if (cnt2 <= 1){
130                     processCommand(commandSemi[cnt]); // 백그라운드가 아니거나 백그라운드 명령어가 한개일때.
131                 } else {
132                     for (int i = 0; i < cnt2; i ++){
133                         if (i == cnt2 - 1 && !isLastBackground) background = FALSE;
134                         processCommand(commandBg[i]);
135                     }
136                 } // 백그라운드 명령어가 두개 이상인 경우.
137             }
138             cnt++;
139             bzero(commandLineArg,COMMANDCNT*ARGSIZE);
140         }
141         bzero(commandLine,INPUTSIZE);
142         bzero(commandSemi,COMMANDCNT*INPUTSIZE);
143         backgroundDone();
144     }
145     /*----- end of while shell loop. -----*/
146 }
147
148 bool parseCommandSemicolon(int *commandLineSemiCnt){
149     bool successParsing = TRUE;
150     bool lastSemi = FALSE;
151     char *tmp = (char *)malloc(sizeof(char)*INPUTSIZE);
152     bzero(tmp,INPUTSIZE);
153     strcpy(tmp,commandLine);
154
155     if (strchr(commandLine,';') && commandLine[0] != ';'){
156         char *token;
```

Line 156, Column 21 — Tab Size: 4 — C

shell.c — UNREGISTERED

```c
        if (commandLine[strlen(commandLine) - 2] == ';') lastSemi = TRUE;
        token = (char *)strtok(commandLine,";");
        while (token != NULL){
            strcpy(commandSemi[*commandLineSemiCnt],trim(token));
            token = (char *)strtok(NULL,";");
            *commandLineSemiCnt += 1;
        }
    } // 입력받은 명령어를 ';' 으로 파싱해줌.
    if (lastSemi) *commandLineSemiCnt -= 1;

    deleteSpaceOrTab(tmp,' ');
    deleteSpaceOrTab(tmp,'\t');
    // 스페이스나 tab 을 모두 제거함.

    if (strstr(tmp,";&") || strstr(tmp,"|&")) {
        printf("-bash: syntax error near unexpected token '&'\n");
        successParsing = FALSE;
    } else if (strstr(tmp,">!;")) {
        printf("-bash: !: event not found\n");
        successParsing = FALSE;
    } else if (strstr(tmp,";|") || strstr(tmp,"&|")) {
        printf("-bash: syntax error near unexpected token '|'\n");
        successParsing = FALSE;
    } else if (strstr(tmp,"&;") || strstr(tmp,">;") || strstr(tmp,">>;") ||
               strstr(tmp,"<;") || strstr(tmp,"<<;")|| strstr(tmp,";;")  || strstr(tmp,"|;")) {
        printf("-bash: syntax error near unexpected token ';'\n");
        successParsing = FALSE;
    }
    // 명령어 상에서의 문법 체크.
    free(tmp);
    return successParsing;
}

void parseCommandArg(char *commandSemi,int *argCnt){
    char *token;
    for (int i = 0; i < strlen(commandSemi); i ++){
        if (commandSemi[i]=='\t') commandSemi[i] = ' ';
    }
    token = (char *)strtok(commandSemi," ");
```

Line 157, Column 74 — Tab Size: 4 — C

shell.c — UNREGISTERED

```c
    while (token != NULL){
        strcpy(commandLineArg[*argCnt],token);
        token = (char *)strtok(NULL," ");
        *argCnt += 1;
    }
    //';' 으로 나누어진 각각의 명령에서 다시 argument 단위로 파싱함.
}

void parseBackground(char *commandSemi,int *cnt2,bool *isLastBackground){

    *isLastBackground = ( commandSemi[strlen(commandSemi) - 1] == '&' ? TRUE : FALSE );
    // 마지막 명령어가 백그라운드 실행인지 확인해줌.
    if (strstr(commandSemi,"&")){
        char *token;
        background = TRUE; // background 실행 여부.
        token = (char *)strtok(commandSemi,"&");
        while (token != NULL){
            strcpy(commandBg[*cnt2],token);
            token = (char *)strtok(NULL,"&");
            *cnt2 += 1;
        }
    } else {
        return ;
    }// 백그라운드 실행 명령어 갯수 세어줌.
}

void printCurrentPath(void){

    char *token,*ptr;
    getcwd(currentPath,PATHSIZE);
    if (strcmp(currentPath,"/")){
        token = strtok(currentPath,"/");
        while (token != NULL){
            ptr = token;
            token = strtok(NULL,"/");
        }
        strcpy(currentPath,ptr);
    } // 현재 경로에서의 마지막 디렉토리만 출력해준다.
    printf("JinHan-MiniShell:%s %s$ ",currentPath,getlogin());
```

Line 196, Column 27 — Tab Size: 4 — C

shell.c — UNREGISTERED

shell.c

```
235     fflush(stdout);
236 }
237
238 void insertHistory(int *historyCnt){
239     if (strcmp(commandLine,"\n")){
240         if (*historyCnt >= HISTORYSIZE - 1) {
241             *historyCnt = HISTORYSIZE - 1;
242             for (int i = 0; i < *historyCnt; i ++){
243                 strcpy(history[i],history[i+1]);
244             }
245             strcpy(history[*historyCnt],commandLine);
246         }
247         else {
248             *historyCnt += 1;
249             strcpy(history[*historyCnt],commandLine);
250         }
251     }// 입력받은 명령어를 Queue 형태의 history 에 저장한다.
252 }
253
254 void showHistory(int *historyCnt){
255     printf("< History >\n");
256     for (int i = 0; i <= *historyCnt; i ++){
257         printf("[%d] : %s\n",i + 1,history[i]);
258     }
259 }
260
261 void processCommand(char *commandSemi){
262     pid_t pid;
263     int pid_status;
264     int argCnt = 0; // argument count.
265     int fd,pipe_cnt = 0;
266
267     deleteSpaceOrTab(commandSemi,'"');
268     bzero(tmpBuffer,INPUTSIZE);
269     strcpy(tmpBuffer,commandSemi);
270
271     parseCommandArg(commandSemi,&argCnt); // 각각 argument 로 파싱함.
272
273     char **argument = NULL;
```

Line 273, Column 28 — Tab Size: 4 — C

shell.c — UNREGISTERED

shell.c

```
274     if (argCnt > 1){
275         argument = (char **)malloc(sizeof(char *)*(argCnt + 1));
276         for (int i = 0; i < argCnt; i ++){
277             *(argument + i) = (char *)malloc(sizeof(char)*ARGSIZE);
278             strcpy(*(argument + i),commandLineArg[i]);
279         }
280         *(argument + argCnt) = NULL;
281     } else {
282         argument = (char **)malloc(sizeof(char *)*2);
283         *(argument) = (char *)malloc(sizeof(char)*ARGSIZE);
284         *(argument + 1) = (char *)malloc(sizeof(char)*ARGSIZE);
285         strcpy(*argument,commandLineArg[0]);
286         *(argument + 1) = NULL;
287     } // 입력받은 명령어를 fork / execvp 에 사용하기 위해 복사한다.
288
289     for (int i = 0; i < argCnt; i ++){
290
291         if (!strcmp(commandLineArg[0],"cd")){
292             return changeDirectory(argCnt);
293         } else if (strstr(commandLineArg[i],">>")){
294             return inoutFileRedirection(2);
295         } else if (strstr(commandLineArg[i],">!")){
296             return inoutFileRedirection(3);
297         } else if (strstr(commandLineArg[i],">")){
298             return inoutFileRedirection(0);
299         } else if (strstr(commandLineArg[i],"<")){
300             return inoutFileRedirection(1);
301         } else if (strstr(commandLineArg[i],"|")){
302             pipe_cnt++;
303         }
304     } // 해당 명령어에 어떠한 type 의 부호가 있는지 판별 후 해당 명령어 처리.
305
306     if (pipe_cnt == 1){
307         return pipeFunction();
308     } else if (pipe_cnt >= 2){
309         return multipipeFunction(pipe_cnt);
310     } // pipe / multiple pipes 인 경우.
311
312     pid = fork();
```

Line 274, Column 21 — Tab Size: 4 — C

shell.c — UNREGISTERED

shell.c

```
313     if (pid < 0) error("Fork Failed");
314     else if (pid == 0) {
315         if (execvp(commandLineArg[0],argument) == -1){
316             printf("-bash: %s: command not found\n",commandLineArg[0]);
317         }
318     } else {
319         if (background) {
320             backgroundPid[backgroundCnt++] = (int)pid;
321             printf("[%d] %d\n",backgroundCnt,pid);
322             // 백그라운드 실행 알림.
323             return ;
324         } else {
325             wait(&pid_status);
326         }
327     }
328 }
329
330 void inoutFileRedirection(int what){
331     pid_t pid;
332     int pip[2];
333     int i = 0,fd,pid_status,n;
334     char *buffer[ARGSIZE],*filename[ARGSIZE];
335     char *token1,*token2,*token3;
336     char *redirection;
337     if (what == 0) redirection = ">";
338     else if (what == 1) redirection = "<";
339     else if (what == 2) redirection = ">>";
340     else if (what == 3) redirection = ">!";
341     // redirection 종류에 따라 token 구분자를 나눈다.
342     token1 = (char *)strtok(tmpBuffer,redirection);
343     token2 = (char *)strtok(NULL,redirection);
344
345     token3 = (char *)strtok(token1," ");
346     while (token3 != NULL){
347         buffer[i++] = token3;
348         token3 = (char *)strtok(NULL," ");
349     }
350     buffer[i] = (char *)'\0';
351     // 명령어를 사용하기 위해 tokenize 된 구문을 다시 스페이스로 파싱함.
```

Line 351, Column 49 — Tab Size: 4 — C

shell.c — UNREGISTERED

shell.c

```
352     i = 0;
353     token3 = (char *)strtok(token2," ");
354     while (token3 != NULL){
355         filename[i++] = token3;
356         token3 = (char *)strtok(NULL," ");
357     }
358     filename[i] = (char *)'\0';
359     // 명령어를 사용하기 위해 tokenize 된 구문을 다시 스페이스로 파싱함.
360
361     if (what == 1){
362         if (pipe(pip) == -1) error("Pipe Error");
363     } // "<" 인 경우 파이프 생성하여 파이프로 해당 data 전송.
364
365     pid = fork();
366     if (pid < 0) error("Fork Failed");
367     else if (pid == 0) {
368         if (what == 0 || what == 2 || what == 3){
369             if (what == 0){
370                 if ( (fd = open(filename[0], O_RDWR | O_CREAT | O_TRUNC, 0644)) == -1 ) error("File Open/Create Failed");
371             } else if (what == 2){
372                 if ( (fd = open(filename[0], O_WRONLY | O_APPEND, 0644)) == -1 ) error("File Open/Create Failed");
373             } else if (what == 3){
374                 if ( ( fd = open(filename[0],O_RDWR | O_CREAT | O_TRUNC, 0644)) == -1 ) error("File Create Failed");
375             }
376             dup2(fd,OUTPUT);
377             close(fd);
378             if (execvp(buffer[0],buffer) == -1) printf("-bash: %s: command not found\n",buffer[0]);
379         } else if (what == 1){
380             char tmp[2];
381             close(pip[0]);
382             dup2(pip[1],INPUT);
383             close(pip[1]);
384             if ( (fd = open(filename[0], O_RDONLY, 0644)) == -1 ) error("File Open Failed");
385             while ( (n = read(fd,tmp,1)) > 0 ) putchar(tmp[0]);
386             if (execvp(buffer[0],buffer) == -1) printf("-bash: %s: command not found\n",buffer[0]);
387             close(fd);
388         }
389     } // 각각 redirection 에 따른 명령어 수행을 처리해준다.
390     else {
```

Line 390, Column 11 — Tab Size: 4 — C

shell.c — UNREGISTERED

shell.c

```
391         if (background) {
392             backgroundPid[backgroundCnt++] = (int)pid;
393             printf("[%d] %d\n",backgroundCnt,pid);
394             // 백그라운드 실행 알림.
395             return ;
396         } else {
397             wait(&pid_status);
398         }
399     }
400 }
401
402 void pipeFunction(void){
403     pid_t pid;
404     int pip[2];
405     int i = 0,fd,pid_status;
406     char *buffer1[ARGSIZE],*buffer2[ARGSIZE];
407     char *token1,*token2,*token3;
408
409     token1 = (char *)strtok(tmpBuffer,"|");
410     token2 = (char *)strtok(NULL,"|");
411
412     token3 = (char *)strtok(token1," ");
413     while (token3 != NULL){
414         buffer1[i++] = token3;
415         token3 = (char *)strtok(NULL," ");
416     }
417     buffer1[i] = (char *)'\0';
418     // 명령어를 사용하기 위해 tokenize 된 구문을 다시 스페이스로 파싱함.
419     i = 0;
420     token3 = (char *)strtok(token2," ");
421     while (token3 != NULL){
422         buffer2[i++] = token3;
423         token3 = (char *)strtok(NULL," ");
424     }
425     buffer2[i] = (char *)'\0';
426     // 명령어를 사용하기 위해 tokenize 된 구문을 다시 스페이스로 파싱함.
427     if (pipe(pip) == -1) error("Pipe Error");
428
429     pid = fork();
```

Line 429, Column 18 — Tab Size: 4 — C

shell.c — UNREGISTERED

shell.c

```
430     if (pid < 0) error("Fork Failed");
431     else if (pid == 0){
432         close(OUTPUT);
433         dup(pip[1]);
434         close(pip[0]);
435         close(pip[1]);
436         if (execvp(buffer1[0], buffer1) == -1 ) printf("-bash: %s: command not found\n",buffer1[0]);
437     } // 첫번째 (앞) 구문 실행.
438     pid = fork();
439     if (pid < 0) error("Fork Failed");
440     else if (pid == 0){
441         close(INPUT);
442         dup(pip[0]);
443         close(pip[0]);
444         close(pip[1]);
445         if (execvp(buffer2[0], buffer2) == -1 ) printf("-bash: %s: command not found\n",buffer2[0]);
446     } // 두번째 (뒤) 구문 실행.
447     close(pip[0]);
448     close(pip[1]);
449     if (background) {
450         backgroundPid[backgroundCnt++] = (int)pid;
451         printf("[%d] %d\n",backgroundCnt,pid);
452         // 백그라운드 실행 알림.
453         return ;
454     } else {
455         wait(&pid_status);
456     }
457 }
458
459 void multipipeFunction(int pipe_cnt){
460     pid_t pid;
461     int pip[2],pid_status;
462     int fd = 0,t_cnt = 0,b_cnt = 0;
463     char *tokenTmp;
464     char *token[pipe_cnt];
465     char *buffer[ARGSIZE];
466
467     tokenTmp = (char *)strtok(tmpBuffer,"|");
468     while (tokenTmp != NULL){
```

Line 429, Column 18 — Tab Size: 4 — C

shell.c — UNREGISTERED

shell.c

```
469         token[t_cnt++] = tokenTmp;
470         tokenTmp = (char *)strtok(NULL,"|");
471     }
472
473     for (int i = 0; i < t_cnt; i ++){
474         bzero(buffer,ARGSIZE);
475         b_cnt = 0;
476         tokenTmp = (char *)strtok(token[i]," ");
477         while (tokenTmp != NULL){
478             buffer[b_cnt++] = tokenTmp;
479             tokenTmp = (char *)strtok(NULL," ");
480         }
481         buffer[b_cnt] = (char *)'\0';
482
483         if (pipe(pip) == -1) error("Pipe Error");
484
485         pid = fork();
486         if (pid < 0) error("Fork Failed");
487         else if (pid == 0){
488             dup2(fd,INPUT);
489             if (i != t_cnt - 1) dup2(pip[1],OUTPUT);
490             close(pip[0]);
491             if (execvp(buffer[0], buffer) == -1 ) printf("-bash: %s: command not found\n",buffer[0]);
492         } else{
493             wait(&pid_status);
494             close(pip[1]);
495             fd = pip[0];
496         }
497     }// pipe 갯수 만큼 돌리면서 서로 서로 pipe 의 입출력을 연결해줌.
498 }
499
500 void changeDirectory(int argCnt){
501     if (argCnt == 1 || !strcmp(commandLineArg[1],"~")) chdir(getenv("HOME"));
502     else {
503         if (chdir(commandLineArg[1]) == -1) {
504             printf("-bash: cd: %s: No such file or directory\n",commandLineArg[1]);
505         }
506     }
507 }
```

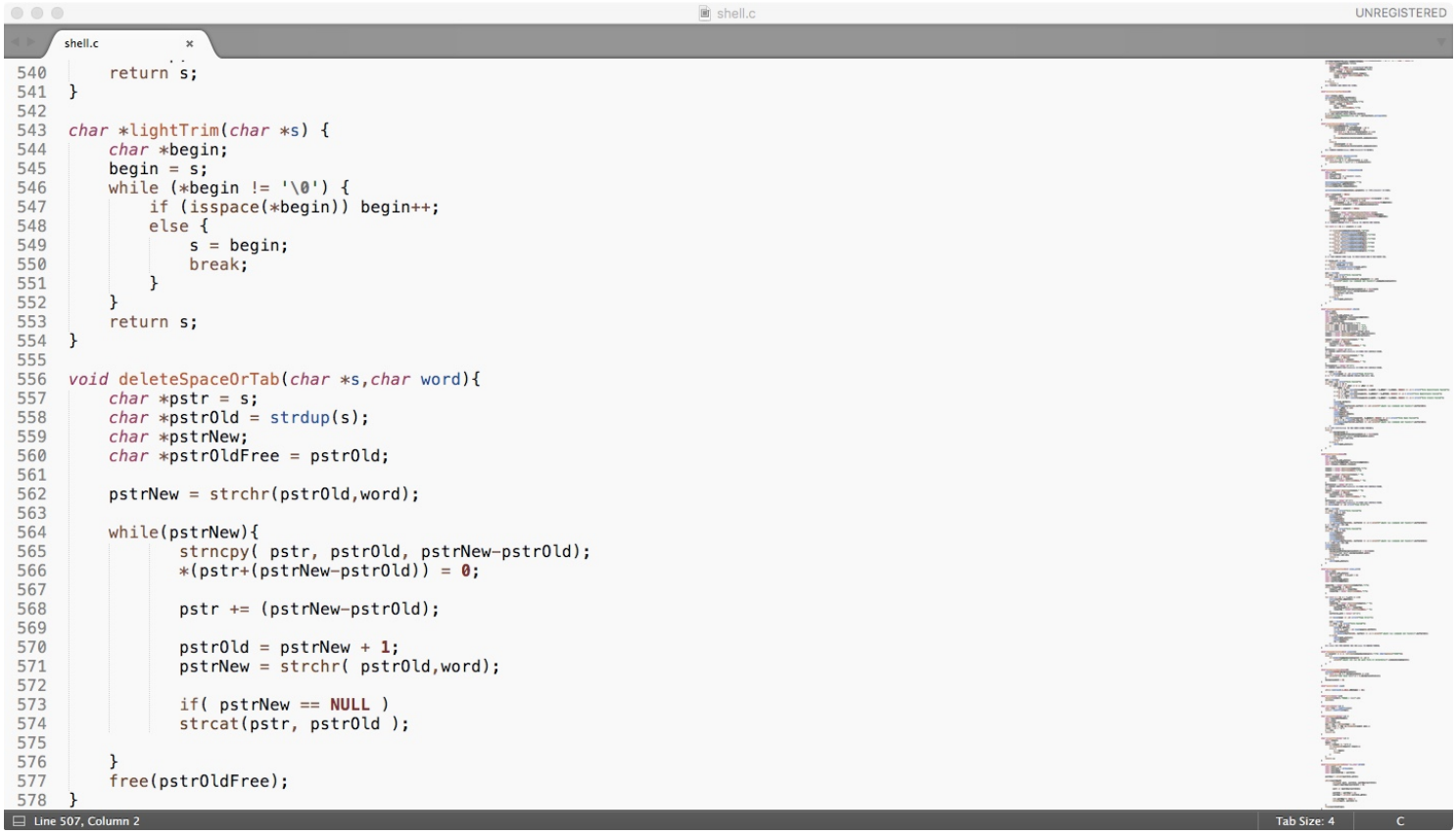
Line 469, Column 35 — Tab Size: 4 — C

shell.c — UNREGISTERED

shell.c

```
508
509 void backgroundDone(void){
510     usleep(100000*backgroundCnt);
511     for (int i = 0; i < backgroundCnt; i ++){
512         printf("[%d] Done %d\n",i + 1,backgroundPid[i]);
513     }
514     backgroundCnt = 0;
515 }
516
517 void handler(int sig){
518
519     while (waitpid(-1,NULL,WNOHANG) > 0);
520 }
521
522 void error(char *s){
523     fprintf(stderr,"ERROR : %s\n",s);
524     exit(1);
525 }
526
527 char *trim(char *s) {
528     char *tmp = lightTrim(s);
529     return rightTrim(tmp);
530 }
531
532 char *rightTrim(char *s) {
533     char tmp[INPUTSIZE2];
534     char *end;
535     strcpy(tmp,s);
536     end = tmp + strlen(tmp) - 1;
537     while (end != tmp && isspace(*end)) end--;
538     *(end + 1) = '\0';
539     s = tmp;
540     return s;
541 }
542
543 char *lightTrim(char *s) {
544     char *begin;
545     begin = s;
546     while (*begin != '\0') {
```

Line 507, Column 2 — Tab Size: 4 — C


shell.c
UNREGISTERED
shell.c
return s;
}
char *lightTrim(char *s) {
char *begin;
begin = s;
while (*begin != '\0') {
if (isspace(*begin)) begin++;
else {
s = begin;
break;
}
}
return s;
}
void deleteSpaceOrTab(char *s,char word){
char *pstr = s;
char *pstrOld = strdup(s);
char *pstrNew;
char *pstrOldFree = pstrOld;
pstrNew = strchr(pstrOld,word);
while(pstrNew){
strncpy(pstr, pstrOld, pstrNew-pstrOld);
*(pstr+(pstrNew-pstrOld)) = 0;
pstr += (pstrNew-pstrOld);
pstrOld = pstrNew + 1;
pstrNew = strchr(pstrOld,word);
if(pstrNew == NULL)
strcat(pstr, pstrOld);
}
free(pstrOldFree);
}
Line 507, Column 2
Tab Size: 4
C

2. Screent Shot

-> foreground execution

```
JinHan-MiniShell:jinhan jinhan$ ls -li -al
total 21484
421207 drwxr-xr-x 19 jinhan jinhan     4096 Dec  2 13:37 .
393217 drwxr-xr-x  3 root   root       4096 Sep  8 00:55 ..
393955 -rw-------  1 jinhan jinhan    21325 Dec  2 12:56 .bash_history
421213 -rw-r--r--  1 jinhan jinhan      220 Sep  8 00:55 .bash_logout
422667 -rw-rw-r--  1 jinhan jinhan       58 Oct 27 17:43 .bash_profile
422108 -rw-r--r--  1 jinhan jinhan     3729 Oct 27 17:45 .bashrc
395227 drwx------ 22 jinhan jinhan     4096 Oct 13 17:04 .cache
394361 drwx------  3 jinhan jinhan     4096 Sep  8 01:05 .compiz
395187 drwx------ 25 jinhan jinhan     4096 Oct 25 22:03 .config
397227 -rw-------  1 jinhan jinhan 21794816 Oct  6 19:20 core
527111 drwx------  3 jinhan jinhan     4096 Oct  8 23:54 .dbus
394837 drwxr-xr-x  5 jinhan jinhan     4096 Dec  1 18:01 Desktop
394819 -rw-r--r--  1 jinhan jinhan       25 Sep  8 00:58 .dmrc
394841 drwxr-xr-x  2 jinhan jinhan     4096 Sep  8 00:58 Documents
394838 drwxr-xr-x  2 jinhan jinhan     4096 Oct 25 21:57 Downloads
421214 -rw-r--r--  1 jinhan jinhan     8980 Sep  8 00:55 examples.desktop
397018 drwx------  4 jinhan jinhan     4096 Dec  2 13:37 .gconf
421530 drwx------  3 jinhan jinhan     4096 Sep 22 17:15 .gnome
411144 -rw-------  1 jinhan jinhan    27030 Dec  2 13:37 .ICEauthority
395260 drwx------  3 jinhan jinhan     4096 Sep  8 00:58 .local
395897 drwx------  4 jinhan jinhan     4096 Sep 22 17:11 .mozilla
394842 drwxr-xr-x  2 jinhan jinhan     4096 Sep  8 00:58 Music
395185 drwxr-xr-x  2 jinhan jinhan     4096 Sep  8 00:58 Pictures
421463 drwx------  3 jinhan jinhan     4096 Sep 22 17:15 .pki
421211 -rw-r--r--  1 jinhan jinhan      675 Sep  8 00:55 .profile
394840 drwxr-xr-x  2 jinhan jinhan     4096 Oct 15 15:22 Public
394839 drwxr-xr-x  2 jinhan jinhan     4096 Sep  8 00:58 Templates
395186 drwxr-xr-x  2 jinhan jinhan     4096 Sep  8 00:58 Videos
430911 -rw-------  1 jinhan jinhan    15121 Nov 24 18:02 .viminfo
421198 -rw-rw-r--  1 jinhan jinhan     1158 Oct  3 23:18 .vimrc
394820 -rw-------  1 jinhan jinhan       51 Dec  2 13:37 .Xauthority
421508 -rw-rw-r--  1 jinhan jinhan      131 Oct  3 23:22 .xinputrc
394368 -rw-------  1 jinhan jinhan       35 Dec  2 13:37 .xsession-errors
394871 -rw-------  1 jinhan jinhan      753 Dec  2 12:56 .xsession-errors.old
JinHan-MiniShell:jinhan jinhan$
```

-> background execution

```
JinHan-MiniShell:2012036901 윤진한 shell jinhan$ ls & ls -li -al
[1] 2866
a.txt  HomeWork2.pdf  MainLoopoftheShell.pdf  shell  shell.c
[1] Done 2866
JinHan-MiniShell:2012036901 윤진한 shell jinhan$ total 375
18 drwxr-xr-x 1 501 dialout    272 Dec  2 12:08 .
 2 drwxrwxrwx 1 501 dialout    340 Dec  2 03:06 ..
28 -rw-r--r-- 1 501 dialout    307 Dec  2 12:03 a.txt
32 -rw-r--r-- 1 501 dialout   6148 Dec  2 03:05 .DS_Store
21 -rw-r--r-- 1 501 dialout 184759 Dec  2 02:40 HomeWork2.pdf
22 -rw-r--r-- 1 501 dialout 149367 Nov  3 14:09 MainLoopoftheShell.pdf
31 -rwxr-xr-x 1 501 dialout  23919 Dec  2 12:08 shell
24 -rw-r--r-- 1 501 dialout  17223 Dec  2 12:05 shell.c
JinHan-MiniShell:2012036901 윤진한 shell jinhan$
```

-> multiple commands separated by semicolons

```
JinHan-MiniShell:2012036901 윤진한 shell jinhan$ ls ; ls -li ; ps
a.txt  HomeWork2.pdf  MainLoopoftheShell.pdf  shell  shell.c
total 368
28 -rw-r--r-- 1 501 dialout    307 Dec  2 12:03 a.txt
21 -rw-r--r-- 1 501 dialout 184759 Dec  2 02:40 HomeWork2.pdf
22 -rw-r--r-- 1 501 dialout 149367 Nov  3 14:09 MainLoopoftheShell.pdf
41 -rwxr-xr-x 1 501 dialout  23919 Dec  2 12:16 shell
24 -rw-r--r-- 1 501 dialout  17239 Dec  2 12:16 shell.c
   PID TTY          TIME CMD
  2506 pts/10   00:00:00 bash
  3020 pts/10   00:00:00 shell
  3024 pts/10   00:00:00 ps
JinHan-MiniShell:2012036901 윤진한 shell jinhan$
```

-> history command

```
JinHan-MiniShell:2012036901 윤진한 shell jinhan$ history
< History >
[1] : clear
[2] : ls -li > a.txt
[3] : cat a.txt
[4] : clear
[5] : history
JinHan-MiniShell:2012036901 윤진한 shell jinhan$
```

-> shell redirection (> , >> , >! , <)

```
JinHan-MiniShell:2012036901 윤진한 shell jinhan$ ls -li > a.txt
JinHan-MiniShell:2012036901 윤진한 shell jinhan$ cat a.txt
total 367
28 -rw-r--r-- 1 501 dialout      0 Dec  2 12:03 a.txt
21 -rw-r--r-- 1 501 dialout 184759 Dec  2 02:40 HomeWork2.pdf
22 -rw-r--r-- 1 501 dialout 149367 Nov  3 14:09 MainLoopoftheShell.pdf
26 -rwxr-xr-x 1 501 dialout  23919 Dec  2 11:57 shell
24 -rw-r--r-- 1 501 dialout  17223 Dec  2 11:57 shell.c
JinHan-MiniShell:2012036901 윤진한 shell jinhan$
```

>

```
JinHan-MiniShell:2012036901 윤진한 shell jinhan$ ls
a.txt  HomeWork2.pdf  MainLoopoftheShell.pdf  shell  shell.c
JinHan-MiniShell:2012036901 윤진한 shell jinhan$ cat a.txt
total 375
18 drwxr-xr-x 1 501 dialout    272 Dec  2 12:16 .
 2 drwxrwxrwx 1 501 dialout    340 Dec  2 03:06 ..
28 -rw-r--r-- 1 501 dialout      0 Dec  2 12:21 a.txt
32 -rw-r--r-- 1 501 dialout   6148 Dec  2 03:05 .DS_Store
21 -rw-r--r-- 1 501 dialout 184759 Dec  2 02:40 HomeWork2.pdf
22 -rw-r--r-- 1 501 dialout 149367 Nov  3 14:09 MainLoopoftheShell.pdf
41 -rwxr-xr-x 1 501 dialout  23919 Dec  2 12:16 shell
24 -rw-r--r-- 1 501 dialout  17239 Dec  2 12:16 shell.c
JinHan-MiniShell:2012036901 윤진한 shell jinhan$ echo "hello World!" >> a.txt
JinHan-MiniShell:2012036901 윤진한 shell jinhan$ cat a.txt
total 375
18 drwxr-xr-x 1 501 dialout    272 Dec  2 12:16 .
 2 drwxrwxrwx 1 501 dialout    340 Dec  2 03:06 ..
28 -rw-r--r-- 1 501 dialout      0 Dec  2 12:21 a.txt
32 -rw-r--r-- 1 501 dialout   6148 Dec  2 03:05 .DS_Store
21 -rw-r--r-- 1 501 dialout 184759 Dec  2 02:40 HomeWork2.pdf
22 -rw-r--r-- 1 501 dialout 149367 Nov  3 14:09 MainLoopoftheShell.pdf
41 -rwxr-xr-x 1 501 dialout  23919 Dec  2 12:16 shell
24 -rw-r--r-- 1 501 dialout  17239 Dec  2 12:16 shell.c
hello World!
JinHan-MiniShell:2012036901 윤진한 shell jinhan$ echo "hello" >! a.txt
JinHan-MiniShell:2012036901 윤진한 shell jinhan$ cat a.txt
hello
JinHan-MiniShell:2012036901 윤진한 shell jinhan$
```

>>

>!

```
JinHan-MiniShell:2012036901 윤진한 shell jinhan$ cat a.txt
total 744
4608363 -rw-r--r--@ 1 JinHan  staff  184759 12  2 02:40 HomeWork2.pdf
3756757 -rw-r--r--@ 1 JinHan  staff  149367 11  3 14:09 MainLoopoftheShell.pdf
4612718 -rw-r--r--  1 JinHan  staff       0 12  2 12:49 a.txt
4613485 -rwxr-xr-x  1 JinHan  staff   19908 12  2 12:49 shell
3889377 -rw-r--r--@ 1 JinHan  staff   17402 12  2 12:49 shell.c
JinHan-MiniShell:2012036901 윤진한 shell jinhan$ head < a.txt
total 744
4608363 -rw-r--r--@ 1 JinHan  staff  184759 12  2 02:40 HomeWork2.pdf
3756757 -rw-r--r--@ 1 JinHan  staff  149367 11  3 14:09 MainLoopoftheShell.pdf
4612718 -rw-r--r--  1 JinHan  staff       0 12  2 12:49 a.txt
4613485 -rwxr-xr-x  1 JinHan  staff   19908 12  2 12:49 shell
3889377 -rw-r--r--@ 1 JinHan  staff   17402 12  2 12:49 shell.c
```

<

-> shell pipe

```
JinHan-MiniShell:bin jinhan$ ls -li -al | more
```

```
131073 drwxr-xr-x  2 root root    4096 Sep  8 00:56 .
     2 drwxr-xr-x 23 root root    4096 Oct  4 16:55 ..
131079 -rwxr-xr-x  1 root root 1021112 Oct  8  2014 bash
131083 -rwxr-xr-x  1 root root   31152 Oct 21  2013 bunzip2
131084 -rwxr-xr-x  1 root root 1918032 Nov 15  2013 busybox
131085 -rwxr-xr-x  1 root root   31152 Oct 21  2013 bzcat
131086 lrwxrwxrwx  1 root root       6 Sep  8 00:50 bzcmp -> bzdiff
131087 -rwxr-xr-x  1 root root    2140 Oct 21  2013 bzdiff
131088 lrwxrwxrwx  1 root root       6 Sep  8 00:50 bzegrep -> bzgrep
131089 -rwxr-xr-x  1 root root    4877 Oct 21  2013 bzexe
131090 lrwxrwxrwx  1 root root       6 Sep  8 00:50 bzfgrep -> bzgrep
131091 -rwxr-xr-x  1 root root    3642 Oct 21  2013 bzgrep
131092 -rwxr-xr-x  1 root root   31152 Oct 21  2013 bzip2
131093 -rwxr-xr-x  1 root root   14480 Oct 21  2013 bzip2recover
131094 lrwxrwxrwx  1 root root       6 Sep  8 00:50 bzless -> bzmore
131095 -rwxr-xr-x  1 root root    1297 Oct 21  2013 bzmore
131096 -rwxr-xr-x  1 root root   47904 Mar 11  2016 cat
131097 -rwxr-xr-x  1 root root   14688 May 23  2013 chacl
131099 -rwxr-xr-x  1 root root   60160 Mar 11  2016 chgrp
131100 -rwxr-xr-x  1 root root   56032 Mar 11  2016 chmod
131101 -rwxr-xr-x  1 root root   60160 Mar 11  2016 chown
131102 -rwxr-xr-x  1 root root   10480 Feb 18  2013 chvt
131103 -rwxr-xr-x  1 root root  130304 Mar 11  2016 cp
131104 -rwxr-xr-x  1 root root  137304 Feb 19  2016 cpio
131105 -rwxr-xr-x  1 root root  121272 Feb 19  2014 dash
131106 -rwxr-xr-x  1 root root   60160 Mar 11  2016 date
131107 -rwxr-xr-x  1 root root   10536 Nov 26  2014 dbus-cleanup-sockets
131108 -rwxr-xr-x  1 root root  434032 Nov 26  2014 dbus-daemon
131109 -rwxr-xr-x  1 root root   10464 Nov 26  2014 dbus-uuidgen
131110 -rwxr-xr-x  1 root root   56136 Mar 11  2016 dd
131113 -rwxr-xr-x  1 root root   97768 Mar 11  2016 df
131114 -rwxr-xr-x  1 root root  110080 Mar 11  2016 dir
131115 -rwxr-xr-x  1 root root   22896 Sep  3  2015 dmesg
131116 lrwxrwxrwx  1 root root       8 Sep  8 00:50 dnsdomainname -> hostname
131117 lrwxrwxrwx  1 root root       8 Sep  8 00:50 domainname -> hostname
131118 -rwxr-xr-x  1 root root   82256 Feb 18  2013 dumpkeys
131119 -rwxr-xr-x  1 root root   31296 Mar 11  2016 echo
131120 -rwxr-xr-x  1 root root   47712 Jul 17  2013 ed
131121 -rwxr-xr-x  1 root root  183696 Jan 19  2014 egrep
131122 -rwxr-xr-x  1 root root   27168 Mar 11  2016 false
131124 -rwxr-xr-x  1 root root   10488 Feb 18  2013 fgconsole
131125 -rwxr-xr-x  1 root root  138352 Jan 19  2014 fgrep
131126 -rwxr-xr-x  1 root root   36144 Sep  3  2015 findmnt
131127 -rwxr-xr-x  1 root root   31864 Nov 30  2012 fuser
131128 -rwsr-xr-x  1 root root   30800 May 15  2015 fusermount
131130 -rwxr-xr-x  1 root root   23688 May 23  2013 getfacl
131131 -rwxr-xr-x  1 root root  191952 Jan 19  2014 grep
--More--
```

-> multiple pipe

```
JinHan-MiniShell:bin jinhan$ ls | grep "^d" | more
dash
date
dbus-cleanup-sockets
dbus-daemon
dbus-uuidgen
dd
df
dir
dmesg
dnsdomainname
domainname
dumpkeys
JinHan-MiniShell:bin jinhan$
```

-> cd command

```
JinHan-MiniShell:2012036901 윤진한 shell jinhan$ cd ..
JinHan-MiniShell:SystemProgram jinhan$ cd ..
JinHan-MiniShell:hgfs jinhan$ cd ~
JinHan-MiniShell:jinhan jinhan$ ls
core  Desktop  Documents  Downloads  examples.desktop  Music  Pictures  Public  Templates  Videos
JinHan-MiniShell:jinhan jinhan$ cd Desktop
JinHan-MiniShell:Desktop jinhan$
```