INSTITUTE OF TECHNICAL EDUCATION AND RESEARCH

(SOA Deemed to be University)

PROJECT REPORT
ON
Custom Shell Implementation

(LINUX OS)

<u>Submitted By:</u>

Name: Raj Vardhan Singh

Registration No.: 2241019502

Branch: CSE

Batch: 13

CODE:

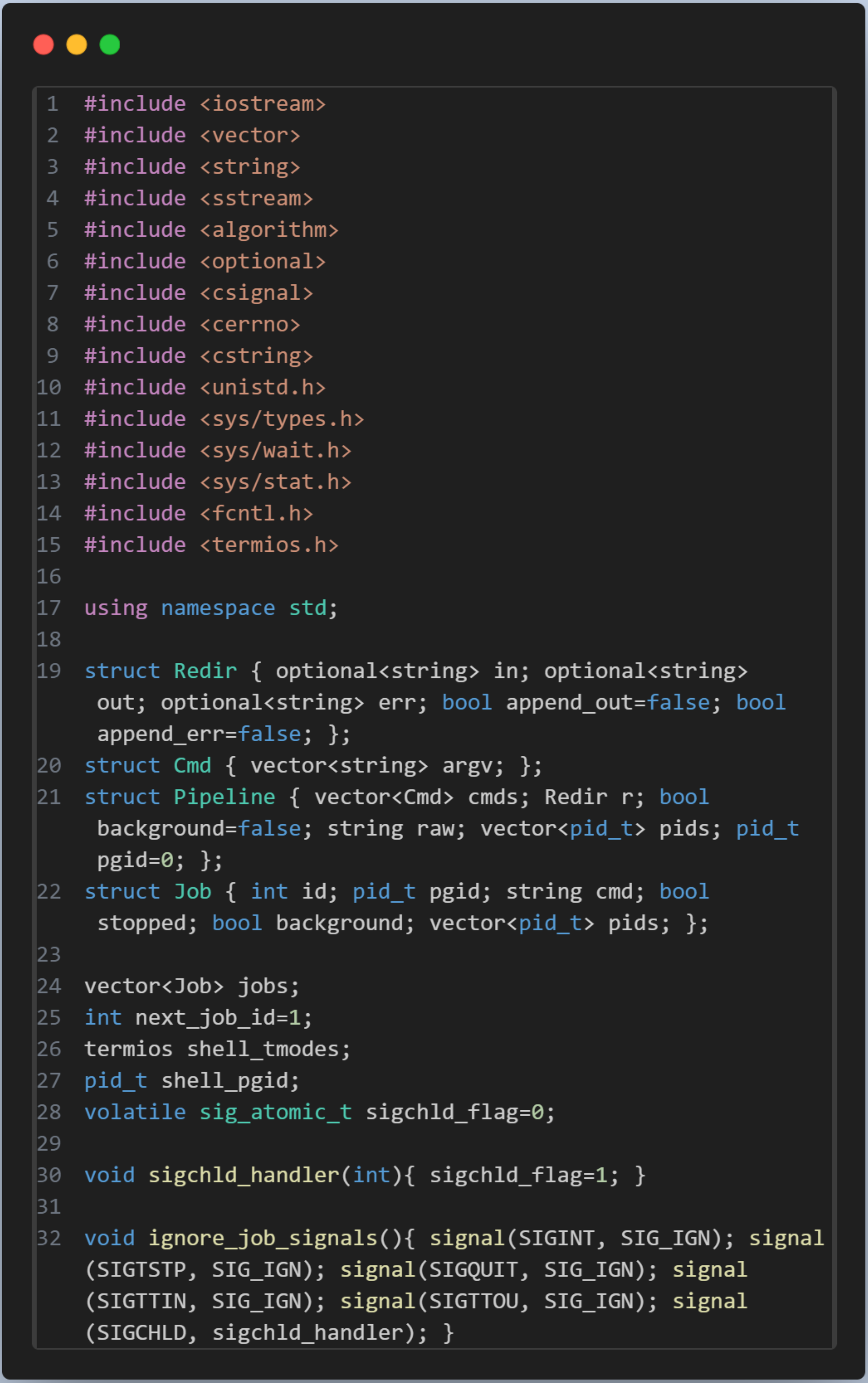
```
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
#include <algorithm>
#include <optional>
#include <csignal>
#include <cerrno>
#include <cstring>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>

using namespace std;

struct Redir { optional<string> in; optional<string> out; optional<string> err; bool
append_out=false; bool append_err=false; };
struct Cmd { vector<string> argv; };
struct Pipeline { vector<Cmd> cmds; Redir r; bool background=false; string raw;
vector<pid_t> pids; pid_t pgid=0; };
struct Job { int id; pid_t pgid; string cmd; bool stopped; bool background; vector<pid_t>
pids; };

vector<Job> jobs;
int next_job_id=1;
termios shell_tmodes;
pid_t shell_pgid;
volatile sig_atomic_t sigchld_flag=0;

void sigchld_handler(int){ sigchld_flag=1; }

void ignore_job_signals(){ signal(SIGINT, SIG_IGN); signal(SIGTSTP, SIG_IGN);
signal(SIGQUIT, SIG_IGN); signal(SIGTTIN, SIG_IGN); signal(SIGTTOU, SIG_IGN);
signal(SIGCHLD, sigchld_handler); }

void restore_job_signals(){ signal(SIGINT, SIG_DFL); signal(SIGTSTP, SIG_DFL);
signal(SIGQUIT, SIG_DFL); signal(SIGTTIN, SIG_DFL); signal(SIGTTOU, SIG_DFL);
signal(SIGCHLD, SIG_DFL); }

vector<string> tokenize(const string& s){
    vector<string> t; string cur; bool sq=false,dq=false,esc=false;
    for(size_t i=0;i<s.size();++i){
        char c=s[i];
        if(esc){ cur.push_back(c); esc=false; continue; }
        if(c=='\\'){ esc=true; continue; }
        if(c=='\''&&!dq){ sq=!sq; continue; }
        if(c=='"'&&!sq){ dq=!dq; continue; }
        if(isspace((unsigned char)c) && !sq && !dq){ if(!cur.empty()){ t.push_back(cur);
cur.clear(); } continue; }
        cur.push_back(c);
    }
    if(!cur.empty()) t.push_back(cur);
    return t;
}

Pipeline parse_line(const string& line){
    Pipeline p; p.raw=line; auto toks=tokenize(line); vector<vector<string>> segments(1);
    for(size_t i=0;i<toks.size();++i){ if(toks[i]=="|") segments.push_back({}); else
segments.back().push_back(toks[i]); }
    if(!segments.empty() && !segments.back().empty() && segments.back().back()=="&"){
p.background=true; segments.back().pop_back(); }
    for(auto& seg:segments){
        Cmd c;
        for(size_t i=0;i<seg.size();++i){
            string tk=seg[i];
            if(tk=="<" && i+1<seg.size()){ p.r.in=seg[++i]; continue; }
            if((tk==">"||tk==">>") && i+1<seg.size()){ p.r.out=seg[++i];
p.r.append_out=(tk==">>"); continue; }
            if((tk=="2>"||tk=="2>>") && i+1<seg.size()){ p.r.err=seg[++i];
p.r.append_err=(tk=="2>>"); continue; }
            c.argv.push_back(tk);
        }
        if(!c.argv.empty()) p.cmds.push_back(c);
    }
    return p;
}

char** build_argv(const vector<string>& v){
    char** a=(char**)calloc(v.size()+1,sizeof(char*));
    for(size_t i=0;i<v.size();++i) a[i]=strdup(v[i].c_str());
    a[v.size()]=nullptr;
    return a;
}

optional<Job*> find_job_by_id(int id){ for(auto& j:jobs) if(j.id==id) return &j; return
nullopt; }

void print_jobs(){ for(auto& j:jobs){ string st=j.stopped?"stopped":"running";
cout<<"["<<j.id<<"] "<<j.pgid<<" "<<st<<" \t"<<j.cmd<<"\n"; } }

void remove_done_jobs(){
    int status;
    for(size_t idx=0; idx<jobs.size();){
        bool any_alive=false;
        for(pid_t pid:jobs[idx].pids){
            if(waitpid(pid,&status,WNOHANG|WUNTRACED|WCONTINUED)>0){
                if(WIFSTOPPED(status)) jobs[idx].stopped=true;
                if(WIFCONTINUED(status)) jobs[idx].stopped=false;
            }
            if(kill(pid,0)==0) any_alive=true;
        }
        if(!any_alive){
            idx=jobs.erase(jobs.begin()+idx)-jobs.begin();
        }else{
            idx++;
        }
    }
}

void reap_children(){
    if(!sigchld_flag) return;
    sigchld_flag=0;
    int status;
    while(true){
        pid_t pid=waitpid(-1,&status,WNOHANG|WUNTRACED|WCONTINUED);
        if(pid<=0) break;
        for(auto& j:jobs){
            if(find(j.pids.begin(),j.pids.end(),pid)!=j.pids.end()){
                if(WIFSTOPPED(status)) j.stopped=true;
                if(WIFCONTINUED(status)) j.stopped=false;
            }
        }
    }
    remove_done_jobs();
}

int open_redir(const string& path,int flags, bool append){
    int f=flags;
    if(((flags & O_WRONLY) || (flags & O_RDWR))
f|=O_CREAT|(append?O_APPEND:O_TRUNC);
    int fd=open(path.c_str(),f,0644);
    return fd;
}

int exec_pipeline(Pipeline& pl){
    size_t n=pl.cmds.size(); if(n==0) return 0;
    vector<int> pipes(max((size_t)0,n-1)*2,-1);
    for(size_t i=0;i+1<n;++i){
        int fds[2];
        if(pipe(fds)==-1){ perror("pipe"); return -1; }
        pipes[2*i]=fds[0];
        pipes[2*i+1]=fds[1];
    }
    pid_t pgid=0; pl.pids.clear();
    for(size_t i=0;i<n;++i){
        pid_t pid=fork();
        if(pid<0){ perror("fork"); return -1; }
        if(pid==0){
            restore_job_signals();
            if(pgid==0) setpgid(0,0); else setpgid(0,pgid);
            if(i>0) dup2(pipes[2*(i-1)], STDIN_FILENO);
            if(i+1<n) dup2(pipes[2*i+1], STDOUT_FILENO);
            for(size_t k=0;k<pipes.size();++k){ if(pipes[k]!=-1) close(pipes[k]); }
            if(pl.r.in && i==0){
                int fd=open_redir(*pl.r.in,O_RDONLY,false);
                if(fd<0){ perror("open"); _exit(1); }
                dup2(fd,STDIN_FILENO); close(fd);
            }
            if(pl.r.out && i==n-1){
                int fd=open_redir(*pl.r.out,O_WRONLY,pl.r.append_out);
                if(fd<0){ perror("open"); _exit(1); }
                dup2(fd,STDOUT_FILENO); close(fd);
            }
            if(pl.r.err && i==n-1){
                int fd=open_redir(*pl.r.err,O_WRONLY,pl.r.append_err);
                if(fd<0){ perror("open"); _exit(1); }
                dup2(fd,STDERR_FILENO); close(fd);
            }
            char** a=build_argv(pl.cmds[i].argv);
            execvp(a[0],a);
            perror("execvp");
            _exit(127);
        }else{
            if(pgid==0) pgid=pid;
            setpgid(pid,pgid);
            pl.pids.push_back(pid);
        }
    }
    for(size_t k=0;k<pipes.size();++k){
        if(pipes[k]!=-1) close(pipes[k]);
    }
    pl.pgid=pgid;
    Job j{next_job_id++, pgid, pl.raw, false, pl.background, pl.pids};
    jobs.push_back(j);
    if(!pl.background){
        tcsetpgrp(STDIN_FILENO, pgid);
        int status; bool stopped=false;
```

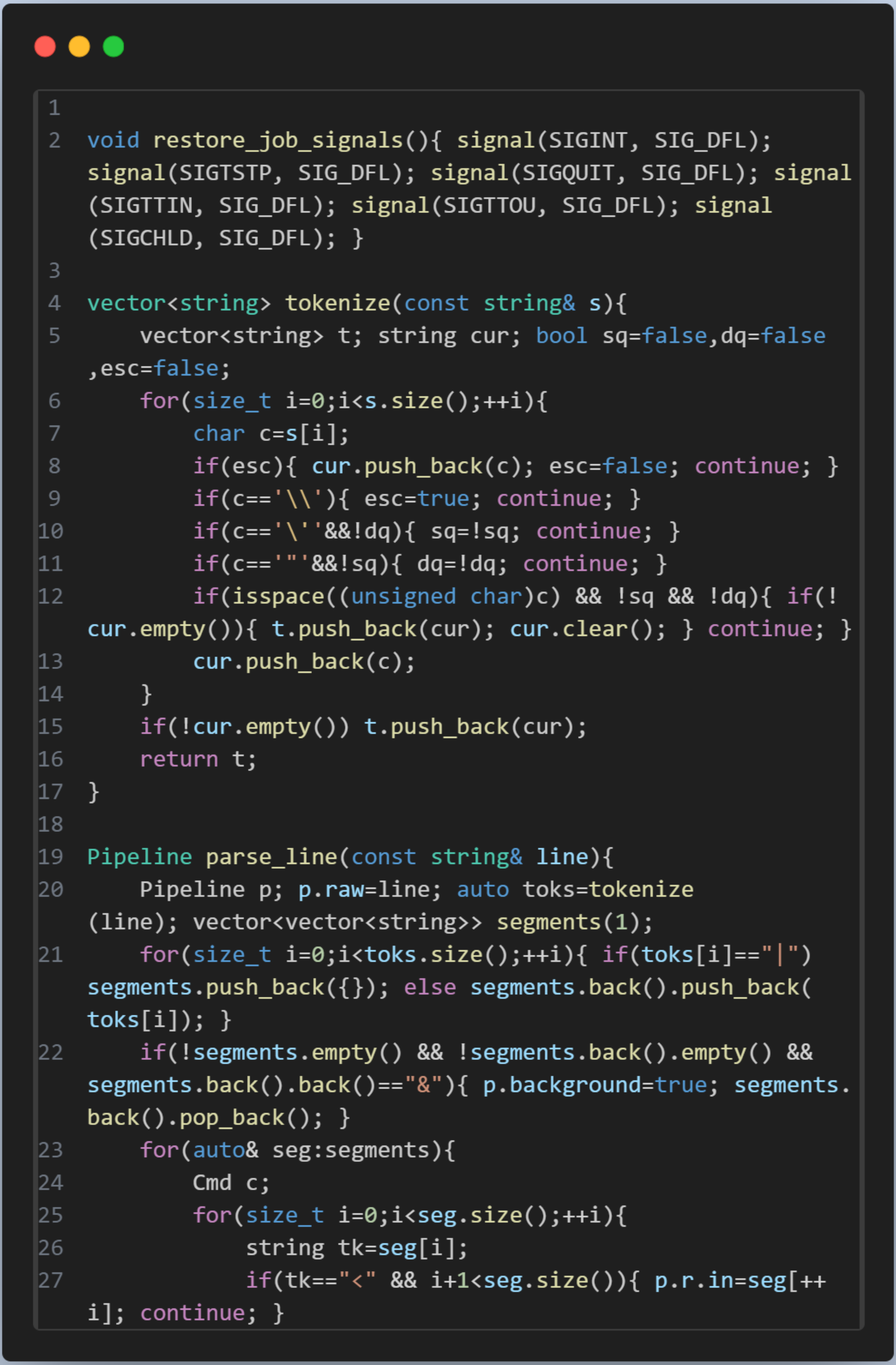
```
        for(pid_t cpid:pl.pids){
            while(true){
                pid_t w=waitpid(cpid,&status,WUNTRACED);
                if(w==-1){
                    if(errno==EINTR) continue;
                    break;
                }
                if(WIFSTOPPED(status)){ stopped=true; break; }
                if(WIFEXITED(status)||WIFSIGNALED(status)) break;
            }
        }
        tcsetpgrp(STDIN_FILENO, shell_pgid);
        tcsetattr(STDIN_FILENO, TCSADRAIN, &shell_tmodes);
        if(!stopped){
            jobs.erase(remove_if(jobs.begin(),jobs.end(),[&](const Job& x){return
x.pgid==pgid;}),jobs.end());
        }
    }else{
        cout<<"["<<jobs.back().id<<"] "<<pgid<<"\n";
    }
    return 0;
}

bool is_number(const string& s){ if(s.empty()) return false; return
all_of(s.begin(),s.end(),::isdigit); }

int builtin_cd(const vector<string>& args){
    const char* path = args.size()>1? args[1].c_str() : getenv("HOME");
    if(!path) path="/";
    if(chdir(path)!=0){ perror("cd"); return 1; }
    return 0;
}

int builtin_pwd(){ char buf[4096]; if(getcwd(buf,sizeof(buf))) cout<<buf<<"\n"; return 0; }

int builtin_jobs(){ reap_children(); print_jobs(); return 0; }

int builtin_fg(const vector<string>& args){
    if(args.size()<2) return 1;
    string a=args[1];
    if(a.size()>0 && a[0]=='%') a=a.substr(1);
    if(!is_number(a)) return 1;
    int id=stoi(a);
    auto oj=find_job_by_id(id); if(!oj) return 1; Job* j=*oj;
    j->background=false;
    tcsetpgrp(STDIN_FILENO, j->pgid);
    kill(-j->pgid, SIGCONT);
    int status;
    for(pid_t p:j->pids){
        while(true){
            pid_t w=waitpid(p,&status,WUNTRACED);
            if(w==-1){
                if(errno==EINTR) continue;
                break;
            }
            if(WIFSTOPPED(status)) { j->stopped=true; break; }
            if(WIFEXITED(status)||WIFSIGNALED(status)) break;
        }
    }
    tcsetpgrp(STDIN_FILENO, shell_pgid);
    tcsetattr(STDIN_FILENO, TCSADRAIN, &shell_tmodes);
    if(!j->stopped){
        jobs.erase(remove_if(jobs.begin(),jobs.end(),[&](const Job& x){return x.id==j-
>id;}),jobs.end());
    }
    return 0;
}

int builtin_bg(const vector<string>& args){
    if(args.size()<2) return 1;
    string a=args[1];
    if(a.size()>0 && a[0]=='%') a=a.substr(1);
    if(!is_number(a)) return 1;
    int id=stoi(a);
    auto oj=find_job_by_id(id); if(!oj) return 1; Job* j=*oj;
    j->background=true; j->stopped=false;
    kill(-j->pgid, SIGCONT);
    cout<<"["<<j->id<<"] "<<j->pgid<<"\n";
    return 0;
}

bool is_builtin(const string& cmd){ static vector<string>
b={"cd","exit","quit","pwd","jobs","fg","bg"}; return find(b.begin(),b.end(),cmd)!=b.end();
}

int run_builtin(const vector<string>& argv){
    string c=argv[0];
    if(c=="cd") return builtin_cd(argv);
    if(c=="pwd") return builtin_pwd();
    if(c=="jobs") return builtin_jobs();
    if(c=="fg") return builtin_fg(argv);
    if(c=="bg") return builtin_bg(argv);
    if(c=="exit"||c=="quit"){ cout<<"bye\n"; exit(0); }
    return 0;
}
```

```
void setup_shell(){
   shell_pgid=getpid();
   setpgid(0,0);
   tcgetattr(STDIN_FILENO,&shell_tmodes);
   tcsetpgrp(STDIN_FILENO, shell_pgid);
}

int main(){
   if(isatty(STDIN_FILENO)) setup_shell();
   ignore_job_signals();
   string line;
   while(true){
      reap_children();
      char cwd[4096];
      if(!getcwd(cwd,sizeof(cwd))){
         perror("getcwd");
         cwd[0]='\0';
      }
      cout<<"mini:"<<cwd<<"$ "<<flush;
      if(!getline(cin,line)){
         cout<<"\n";
         break;
      }
      if(line.size()==0) continue;
      Pipeline pl=parse_line(line);
      if(pl.cmds.empty()) continue;
      if(is_builtin(pl.cmds[0].argv[0]) && pl.cmds.size()==1 && !pl.r.in && !pl.r.out &&
!pl.r.err){
         run_builtin(pl.cmds[0].argv);
         continue;
      }
      exec_pipeline(pl);
   }
   return 0;
}
```

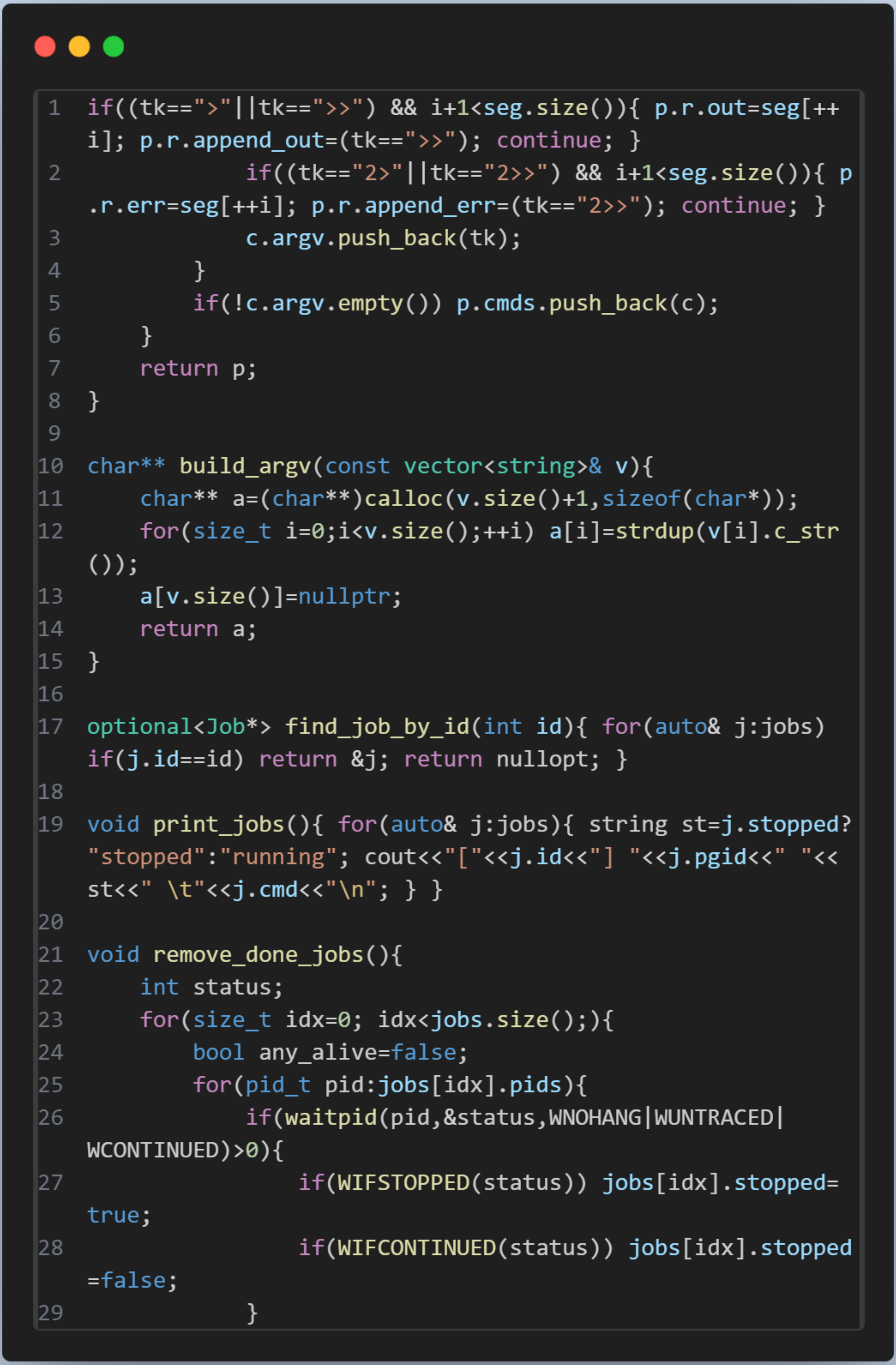
Screenshots

Code:

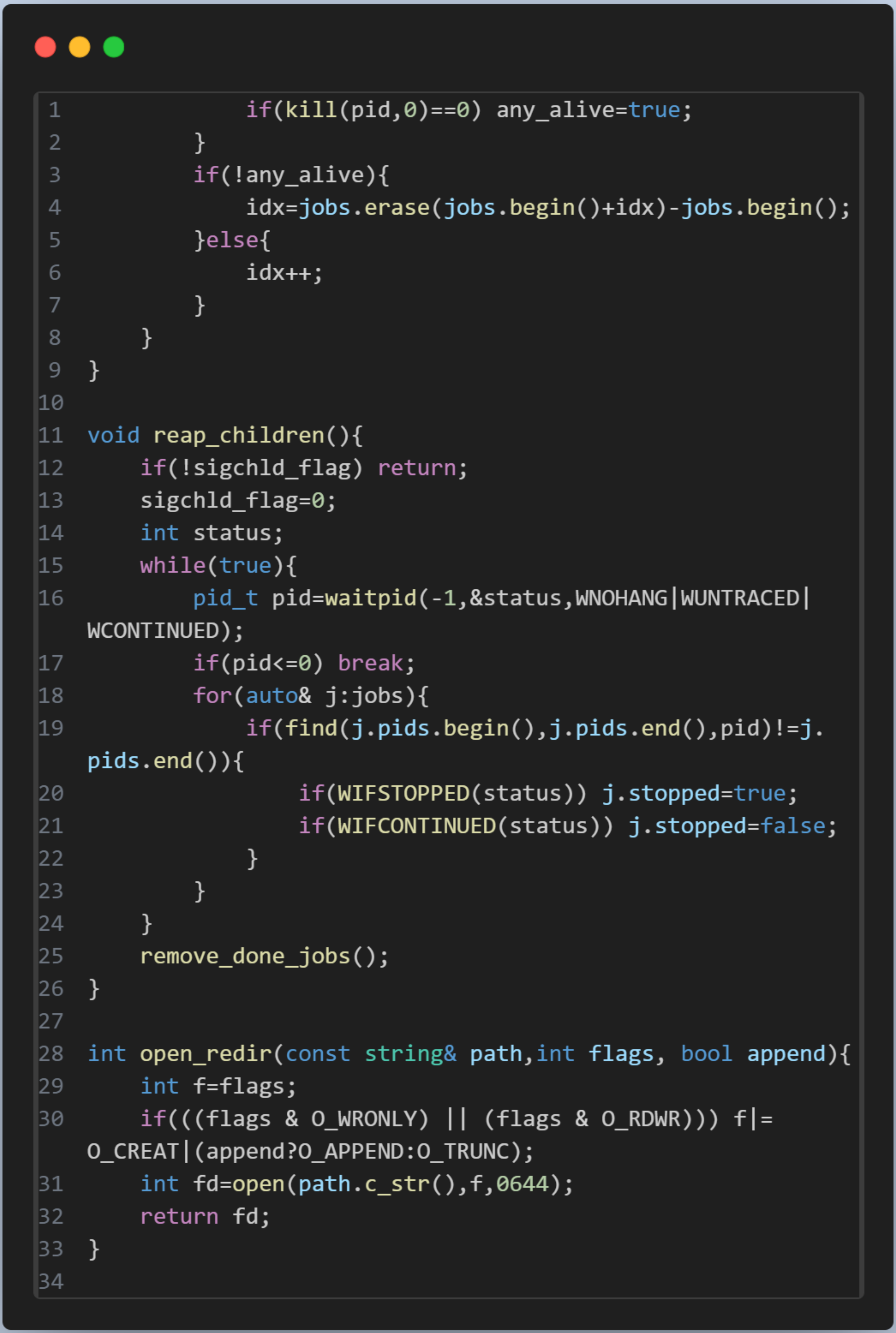

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
#include <algorithm>
#include <optional>
#include <csignal>
#include <cerrno>
#include <cstring>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>

using namespace std;

struct Redir { optional<string> in; optional<string> out; optional<string> err; bool append_out=false; bool append_err=false; };
struct Cmd { vector<string> argv; };
struct Pipeline { vector<Cmd> cmds; Redir r; bool background=false; string raw; vector<pid_t> pids; pid_t pgid=0; };
struct Job { int id; pid_t pgid; string cmd; bool stopped; bool background; vector<pid_t> pids; };

vector<Job> jobs;
int next_job_id=1;
termios shell_tmodes;
pid_t shell_pgid;
volatile sig_atomic_t sigchld_flag=0;

void sigchld_handler(int){ sigchld_flag=1; }

void ignore_job_signals(){ signal(SIGINT, SIG_IGN); signal(SIGTSTP, SIG_IGN); signal(SIGQUIT, SIG_IGN); signal(SIGTTIN, SIG_IGN); signal(SIGTTOU, SIG_IGN); signal(SIGCHLD, sigchld_handler); }
```



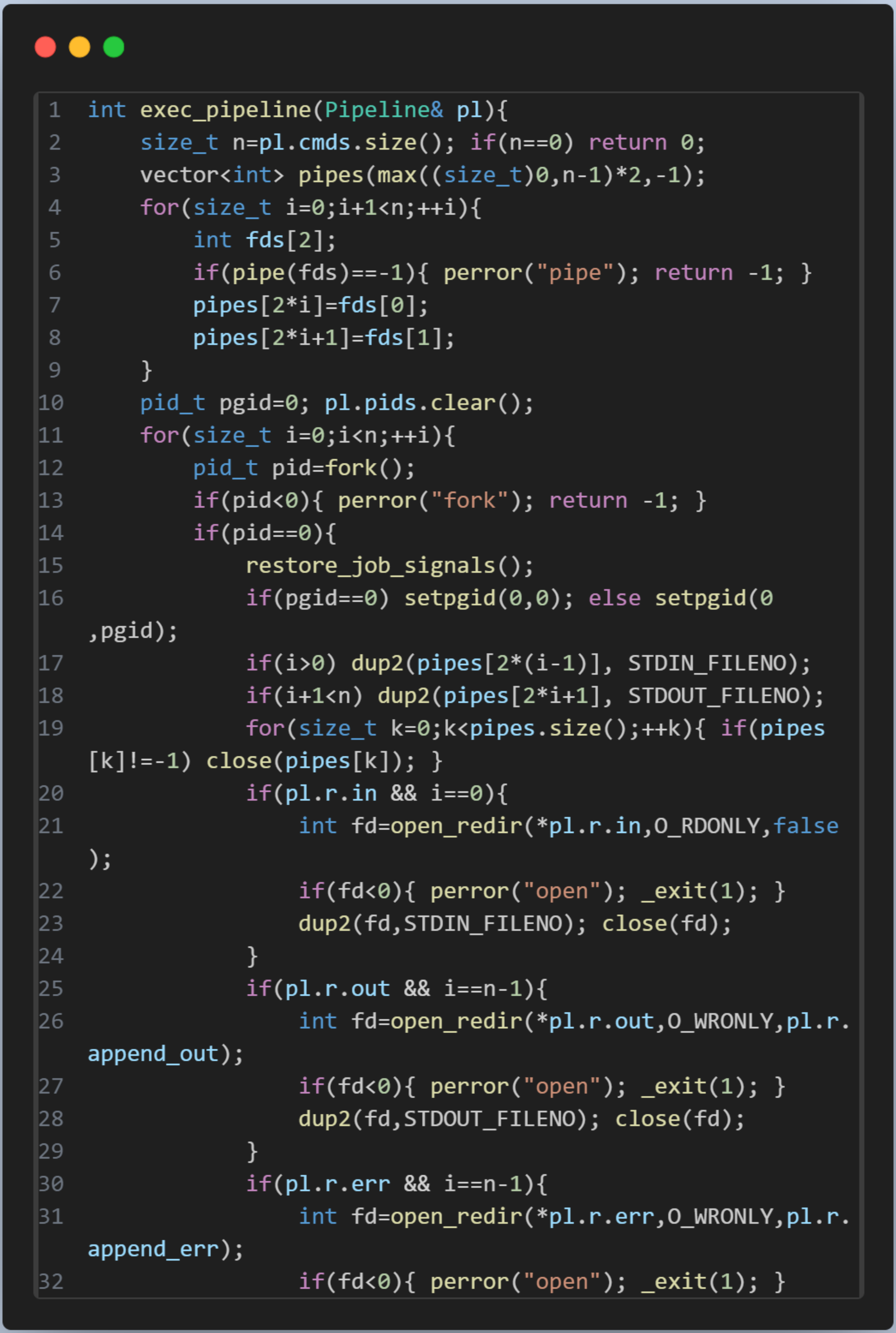
```cpp

void restore_job_signals(){ signal(SIGINT, SIG_DFL); signal(SIGTSTP, SIG_DFL); signal(SIGQUIT, SIG_DFL); signal(SIGTTIN, SIG_DFL); signal(SIGTTOU, SIG_DFL); signal(SIGCHLD, SIG_DFL); }

vector<string> tokenize(const string& s){
    vector<string> t; string cur; bool sq=false,dq=false,esc=false;
    for(size_t i=0;i<s.size();++i){
        char c=s[i];
        if(esc){ cur.push_back(c); esc=false; continue; }
        if(c=='\\'){ esc=true; continue; }
        if(c=='\''&&!dq){ sq=!sq; continue; }
        if(c=='"'&&!sq){ dq=!dq; continue; }
        if(isspace((unsigned char)c) && !sq && !dq){ if(!cur.empty()){ t.push_back(cur); cur.clear(); } continue; }
        cur.push_back(c);
    }
    if(!cur.empty()) t.push_back(cur);
    return t;
}

Pipeline parse_line(const string& line){
    Pipeline p; p.raw=line; auto toks=tokenize(line); vector<vector<string>> segments(1);
    for(size_t i=0;i<toks.size();++i){ if(toks[i]=="|") segments.push_back({}); else segments.back().push_back(toks[i]); }
    if(!segments.empty() && !segments.back().empty() && segments.back().back()=="&"){ p.background=true; segments.back().pop_back(); }
    for(auto& seg:segments){
        Cmd c;
        for(size_t i=0;i<seg.size();++i){
            string tk=seg[i];
            if(tk=="<" && i+1<seg.size()){ p.r.in=seg[++i]; continue; }
```



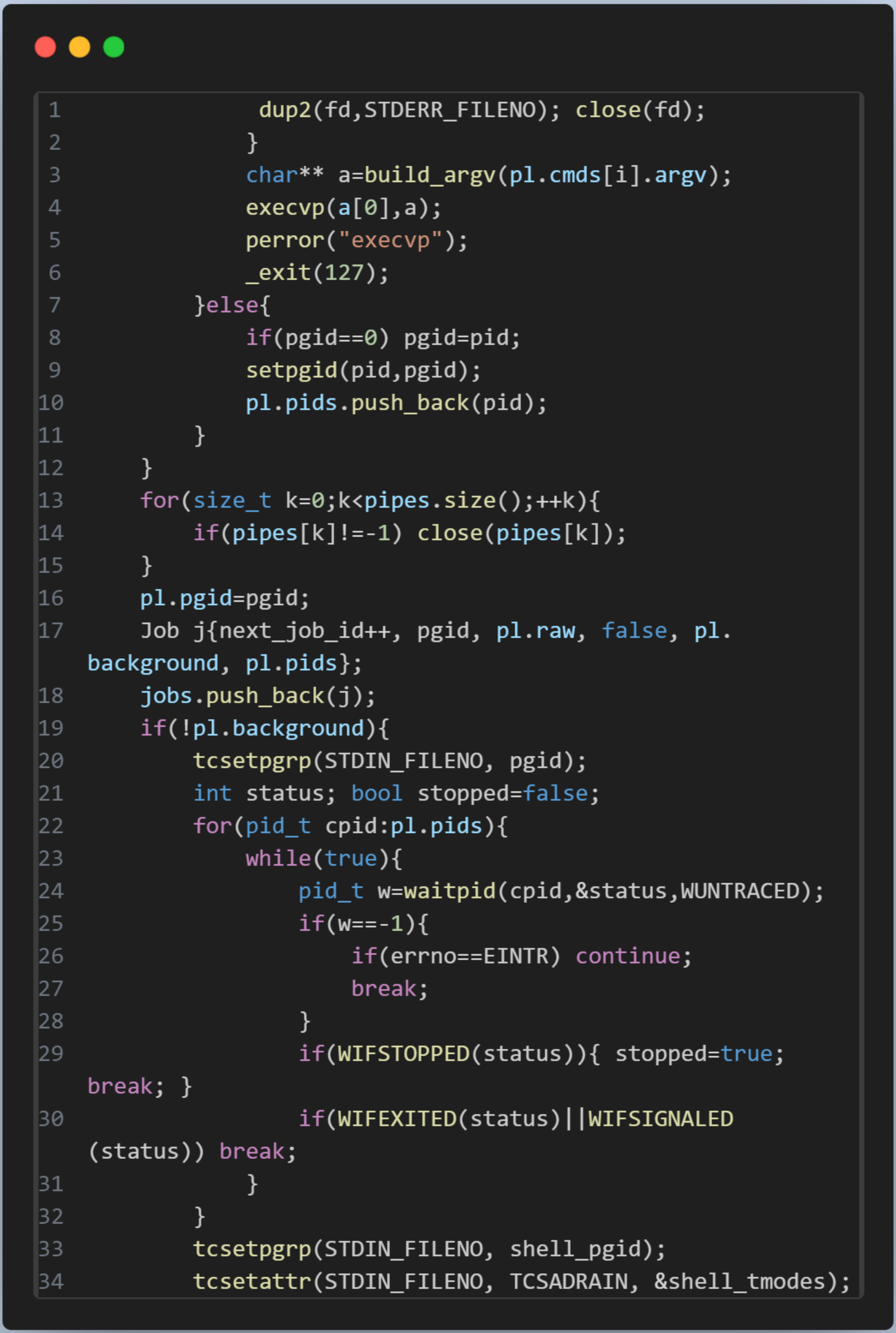
```cpp
if((tk==">"||tk==">>") && i+1<seg.size()){ p.r.out=seg[++i]; p.r.append_out=(tk==">>"); continue; }
            if((tk=="2>"||tk=="2>>") && i+1<seg.size()){ p.r.err=seg[++i]; p.r.append_err=(tk=="2>>"); continue; }
            c.argv.push_back(tk);
        }
        if(!c.argv.empty()) p.cmds.push_back(c);
    }
    return p;
}

char** build_argv(const vector<string>& v){
    char** a=(char**)calloc(v.size()+1,sizeof(char*));
    for(size_t i=0;i<v.size();++i) a[i]=strdup(v[i].c_str());
    a[v.size()]=nullptr;
    return a;
}

optional<Job*> find_job_by_id(int id){ for(auto& j:jobs) if(j.id==id) return &j; return nullopt; }

void print_jobs(){ for(auto& j:jobs){ string st=j.stopped?"stopped":"running"; cout<<"["<<j.id<<"] "<<j.pgid<<" "<<st<<" \t"<<j.cmd<<"\n"; } }

void remove_done_jobs(){
    int status;
    for(size_t idx=0; idx<jobs.size();){
        bool any_alive=false;
        for(pid_t pid:jobs[idx].pids){
            if(waitpid(pid,&status,WNOHANG|WUNTRACED|WCONTINUED)>0){
                if(WIFSTOPPED(status)) jobs[idx].stopped=true;
                if(WIFCONTINUED(status)) jobs[idx].stopped=false;
            }
```



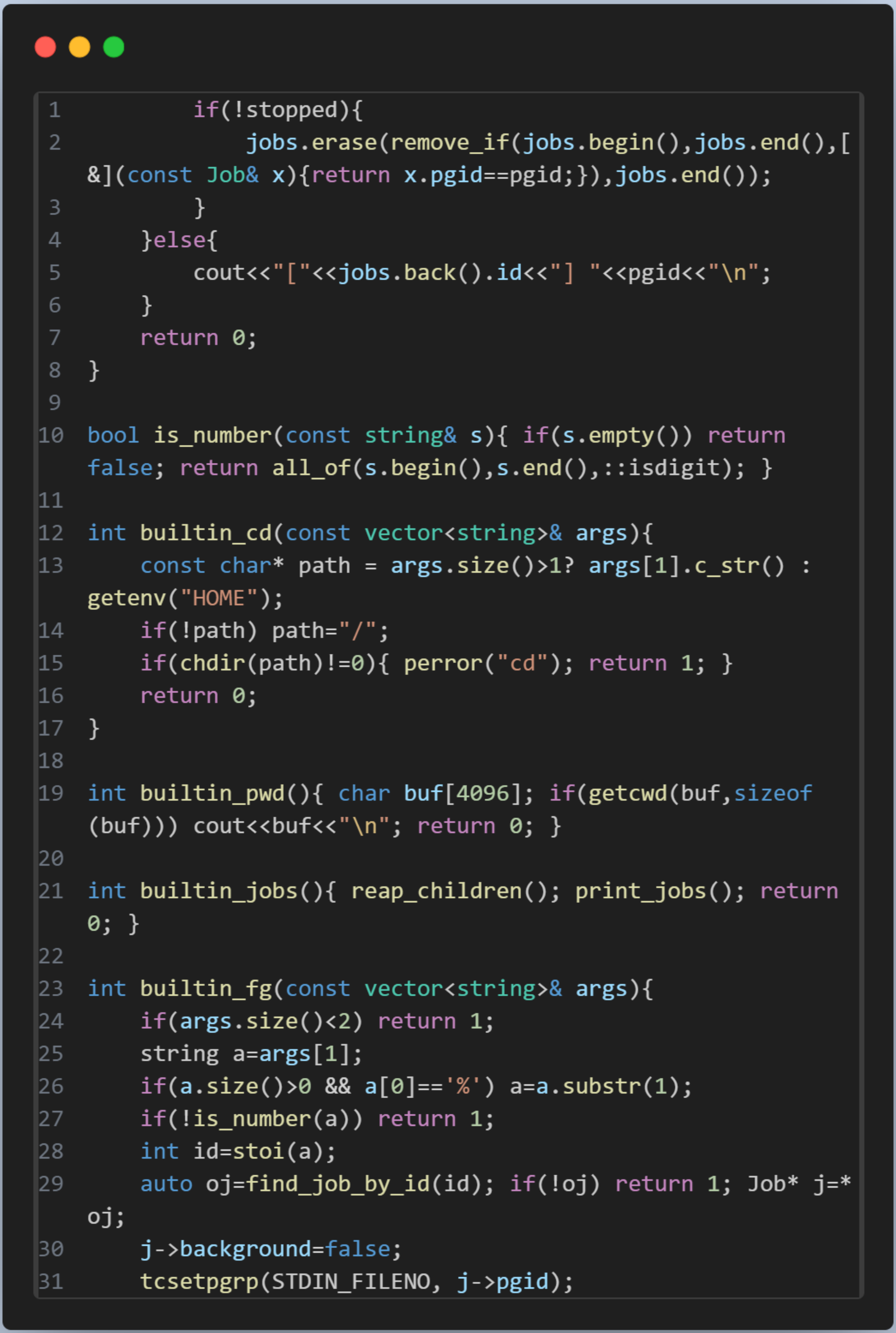
```cpp
            if(kill(pid,0)==0) any_alive=true;
        }
        if(!any_alive){
            idx=jobs.erase(jobs.begin()+idx)-jobs.begin();
        }else{
            idx++;
        }
    }
}

void reap_children(){
    if(!sigchld_flag) return;
    sigchld_flag=0;
    int status;
    while(true){
        pid_t pid=waitpid(-1,&status,WNOHANG|WUNTRACED|WCONTINUED);
        if(pid<=0) break;
        for(auto& j:jobs){
            if(find(j.pids.begin(),j.pids.end(),pid)!=j.pids.end()){
                if(WIFSTOPPED(status)) j.stopped=true;
                if(WIFCONTINUED(status)) j.stopped=false;
            }
        }
    }
    remove_done_jobs();
}

int open_redir(const string& path,int flags, bool append){
    int f=flags;
    if(((flags & O_WRONLY) || (flags & O_RDWR))) f|=O_CREAT|(append?O_APPEND:O_TRUNC);
    int fd=open(path.c_str(),f,0644);
    return fd;
}
```



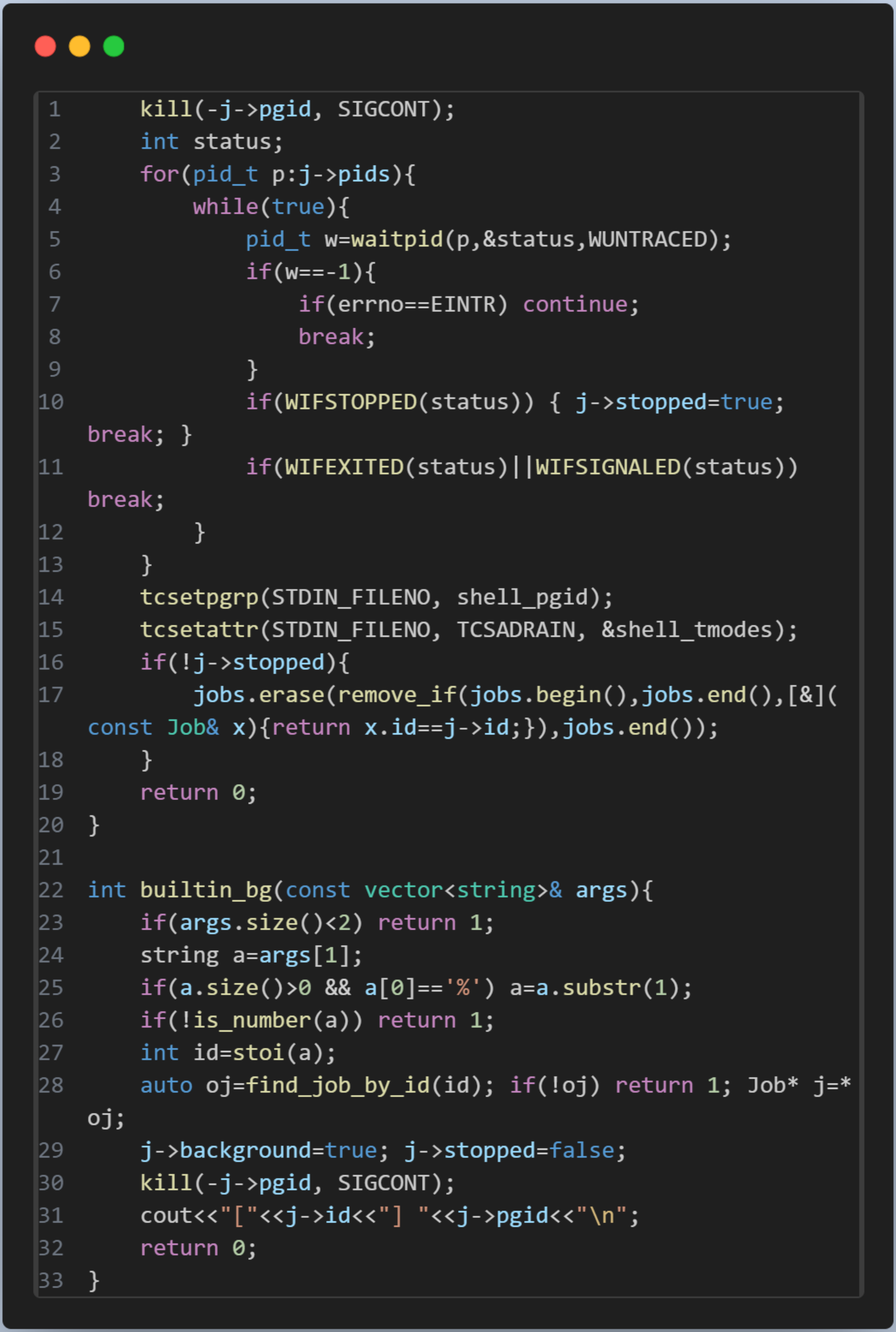
```cpp
int exec_pipeline(Pipeline& pl){
    size_t n=pl.cmds.size(); if(n==0) return 0;
    vector<int> pipes(max((size_t)0,n-1)*2,-1);
    for(size_t i=0;i+1<n;++i){
        int fds[2];
        if(pipe(fds)==-1){ perror("pipe"); return -1; }
        pipes[2*i]=fds[0];
        pipes[2*i+1]=fds[1];
    }
    pid_t pgid=0; pl.pids.clear();
    for(size_t i=0;i<n;++i){
        pid_t pid=fork();
        if(pid<0){ perror("fork"); return -1; }
        if(pid==0){
            restore_job_signals();
            if(pgid==0) setpgid(0,0); else setpgid(0,pgid);
            if(i>0) dup2(pipes[2*(i-1)], STDIN_FILENO);
            if(i+1<n) dup2(pipes[2*i+1], STDOUT_FILENO);
            for(size_t k=0;k<pipes.size();++k){ if(pipes[k]!=-1) close(pipes[k]); }
            if(pl.r.in && i==0){
                int fd=open_redir(*pl.r.in,O_RDONLY,false);
                if(fd<0){ perror("open"); _exit(1); }
                dup2(fd,STDIN_FILENO); close(fd);
            }
            if(pl.r.out && i==n-1){
                int fd=open_redir(*pl.r.out,O_WRONLY,pl.r.append_out);
                if(fd<0){ perror("open"); _exit(1); }
                dup2(fd,STDOUT_FILENO); close(fd);
            }
            if(pl.r.err && i==n-1){
                int fd=open_redir(*pl.r.err,O_WRONLY,pl.r.append_err);
                if(fd<0){ perror("open"); _exit(1); }
```



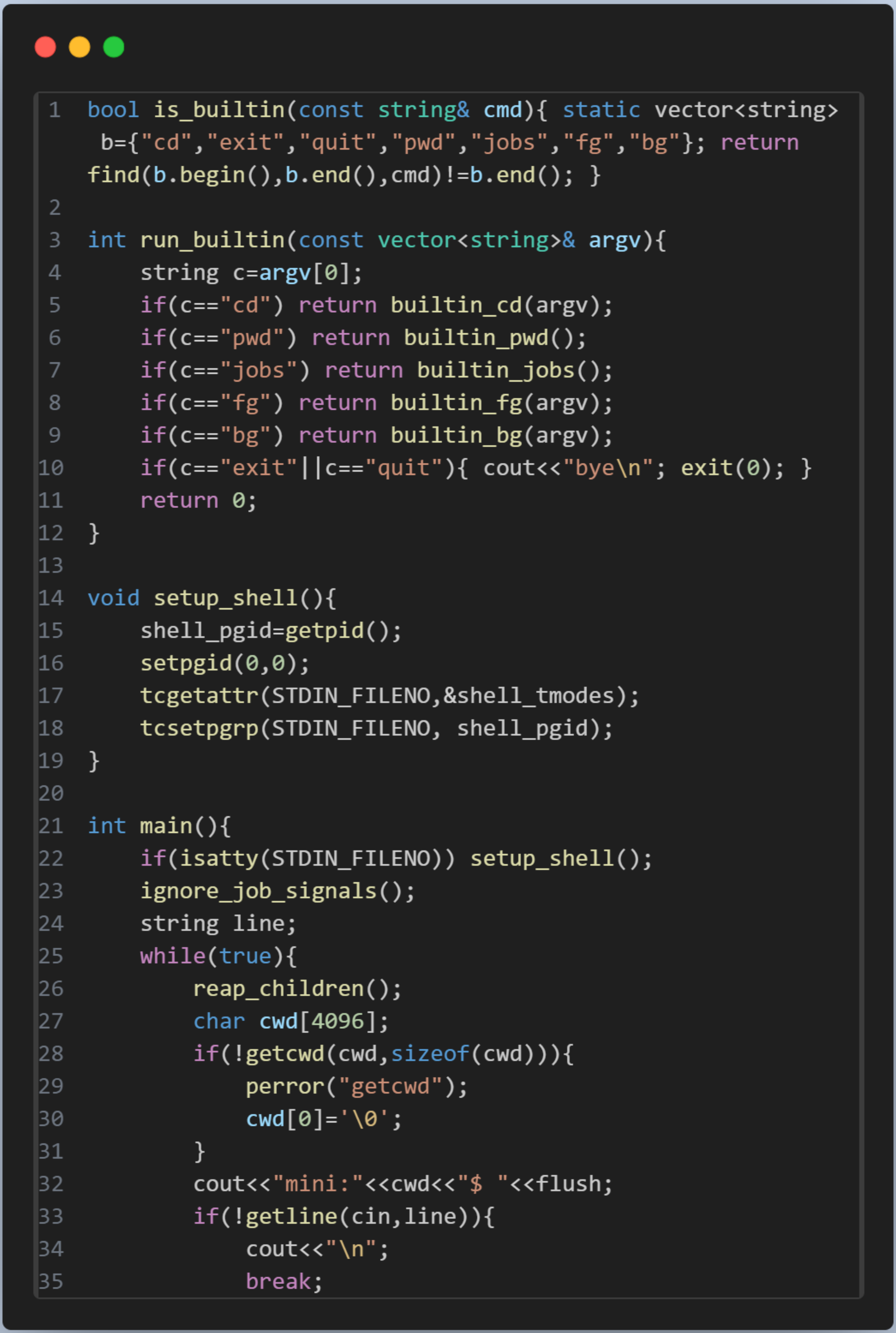
```
             dup2(fd,STDERR_FILENO); close(fd);
            }
            char** a=build_argv(pl.cmds[i].argv);
            execvp(a[0],a);
            perror("execvp");
            _exit(127);
        }else{
            if(pgid==0) pgid=pid;
            setpgid(pid,pgid);
            pl.pids.push_back(pid);
        }
    }
    for(size_t k=0;k<pipes.size();++k){
        if(pipes[k]!=-1) close(pipes[k]);
    }
    pl.pgid=pgid;
    Job j{next_job_id++, pgid, pl.raw, false, pl.background, pl.pids};
    jobs.push_back(j);
    if(!pl.background){
        tcsetpgrp(STDIN_FILENO, pgid);
        int status; bool stopped=false;
        for(pid_t cpid:pl.pids){
            while(true){
                pid_t w=waitpid(cpid,&status,WUNTRACED);
                if(w==-1){
                    if(errno==EINTR) continue;
                    break;
                }
                if(WIFSTOPPED(status)){ stopped=true; break; }
                if(WIFEXITED(status)||WIFSIGNALED(status)) break;
            }
        }
        tcsetpgrp(STDIN_FILENO, shell_pgid);
        tcsetattr(STDIN_FILENO, TCSADRAIN, &shell_tmodes);
```

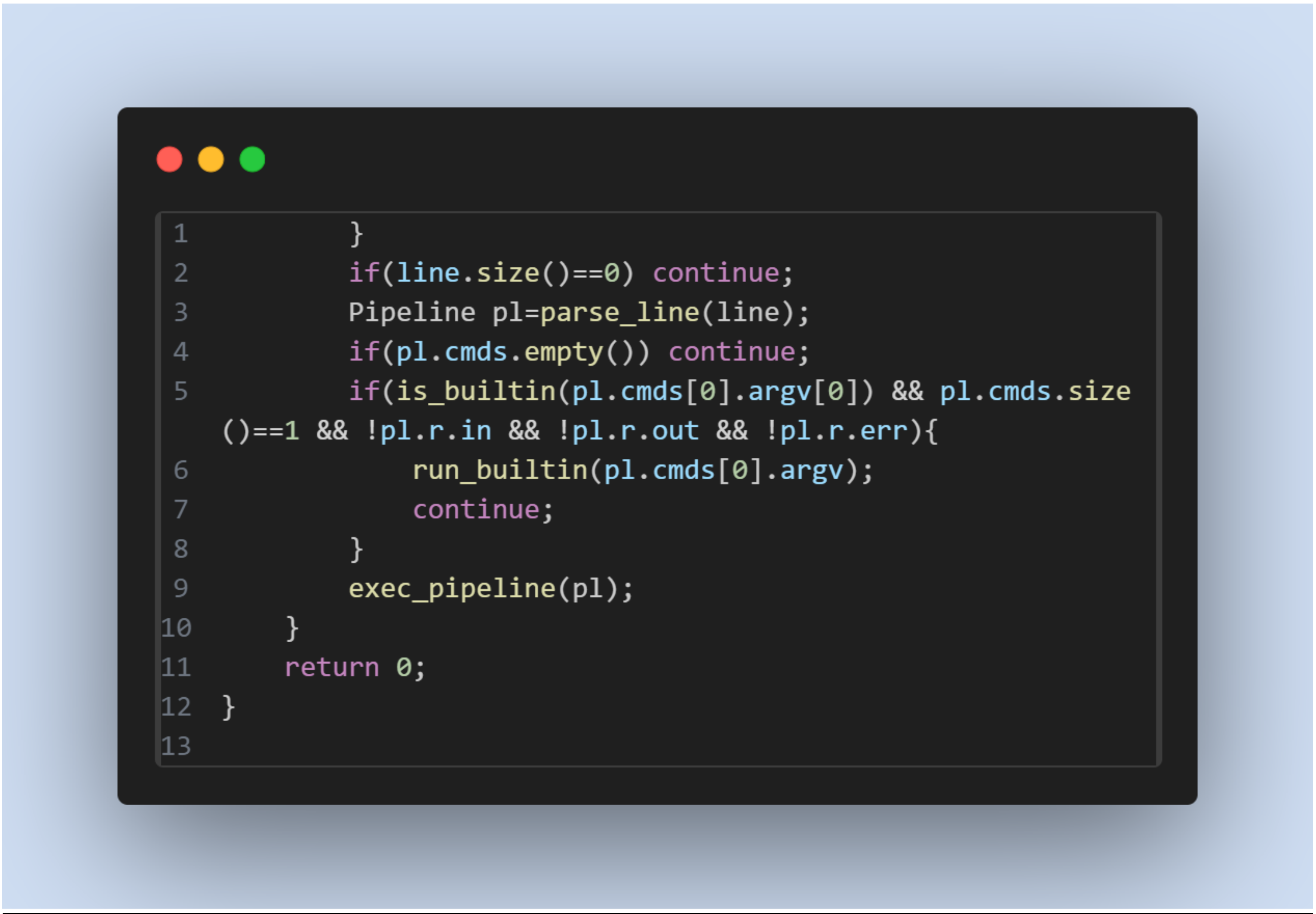


```cpp
        if(!stopped){
            jobs.erase(remove_if(jobs.begin(),jobs.end(),[&](const Job& x){return x.pgid==pgid;}),jobs.end());
        }
    }else{
        cout<<"["<<jobs.back().id<<"] "<<pgid<<"\n";
    }
    return 0;
}

bool is_number(const string& s){ if(s.empty()) return false; return all_of(s.begin(),s.end(),::isdigit); }

int builtin_cd(const vector<string>& args){
    const char* path = args.size()>1? args[1].c_str() : getenv("HOME");
    if(!path) path="/";
    if(chdir(path)!=0){ perror("cd"); return 1; }
    return 0;
}

int builtin_pwd(){ char buf[4096]; if(getcwd(buf,sizeof(buf))) cout<<buf<<"\n"; return 0; }

int builtin_jobs(){ reap_children(); print_jobs(); return 0; }

int builtin_fg(const vector<string>& args){
    if(args.size()<2) return 1;
    string a=args[1];
    if(a.size()>0 && a[0]=='%') a=a.substr(1);
    if(!is_number(a)) return 1;
    int id=stoi(a);
    auto oj=find_job_by_id(id); if(!oj) return 1; Job* j=*oj;
    j->background=false;
    tcsetpgrp(STDIN_FILENO, j->pgid);
```



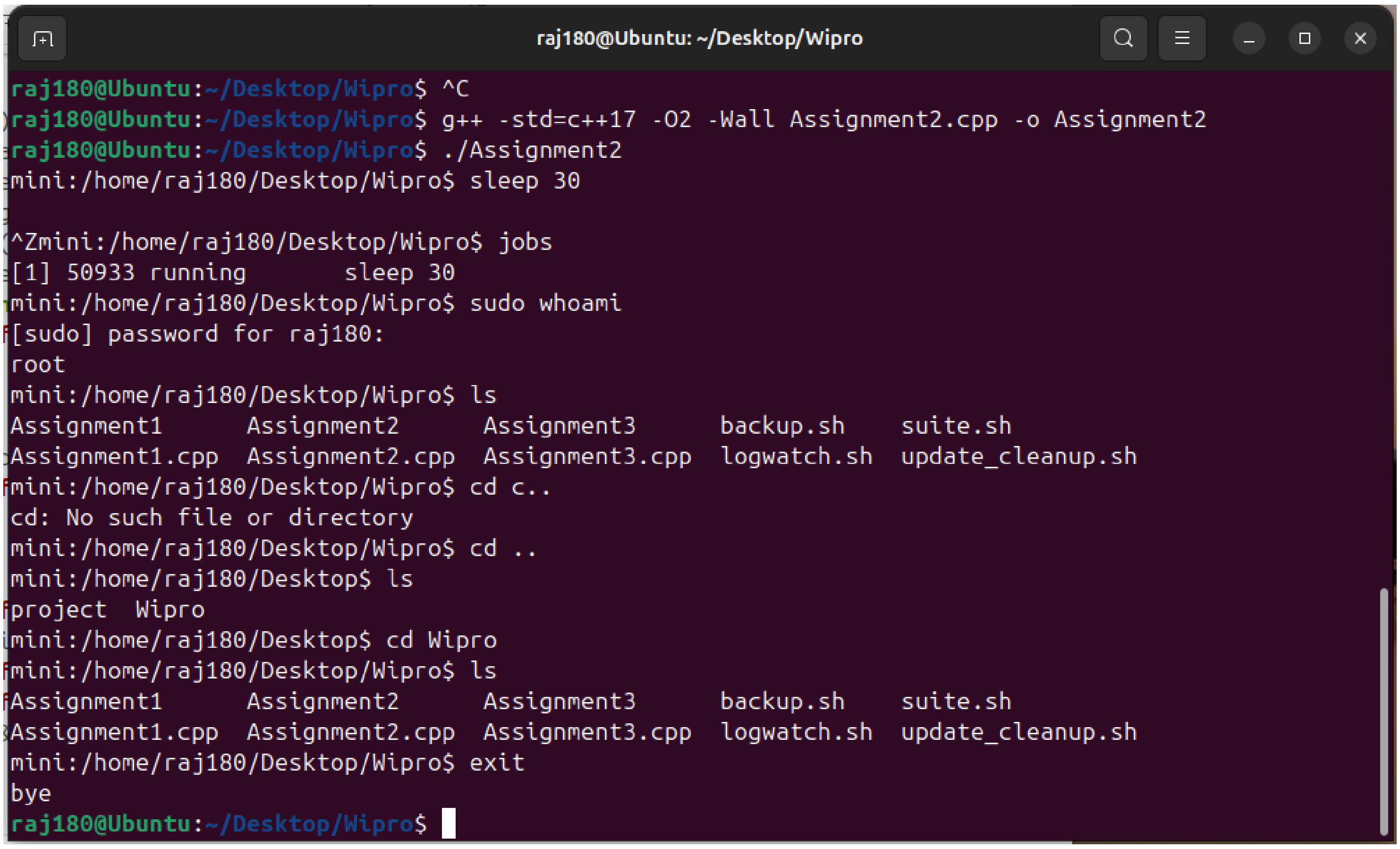
```
    kill(-j->pgid, SIGCONT);
    int status;
    for(pid_t p:j->pids){
        while(true){
            pid_t w=waitpid(p,&status,WUNTRACED);
            if(w==-1){
                if(errno==EINTR) continue;
                break;
            }
            if(WIFSTOPPED(status)) { j->stopped=true; break; }
            if(WIFEXITED(status)||WIFSIGNALED(status)) break;
        }
    }
    tcsetpgrp(STDIN_FILENO, shell_pgid);
    tcsetattr(STDIN_FILENO, TCSADRAIN, &shell_tmodes);
    if(!j->stopped){
        jobs.erase(remove_if(jobs.begin(),jobs.end(),[&](const Job& x){return x.id==j->id;}),jobs.end());
    }
    return 0;
}

int builtin_bg(const vector<string>& args){
    if(args.size()<2) return 1;
    string a=args[1];
    if(a.size()>0 && a[0]=='%') a=a.substr(1);
    if(!is_number(a)) return 1;
    int id=stoi(a);
    auto oj=find_job_by_id(id); if(!oj) return 1; Job* j=*oj;
    j->background=true; j->stopped=false;
    kill(-j->pgid, SIGCONT);
    cout<<"["<<j->id<<"] "<<j->pgid<<"\n";
    return 0;
}
```


```cpp
bool is_builtin(const string& cmd){ static vector<string>
 b={"cd","exit","quit","pwd","jobs","fg","bg"}; return
find(b.begin(),b.end(),cmd)!=b.end(); }

int run_builtin(const vector<string>& argv){
    string c=argv[0];
    if(c=="cd") return builtin_cd(argv);
    if(c=="pwd") return builtin_pwd();
    if(c=="jobs") return builtin_jobs();
    if(c=="fg") return builtin_fg(argv);
    if(c=="bg") return builtin_bg(argv);
    if(c=="exit"||c=="quit"){ cout<<"bye\n"; exit(0); }
    return 0;
}

void setup_shell(){
    shell_pgid=getpid();
    setpgid(0,0);
    tcgetattr(STDIN_FILENO,&shell_tmodes);
    tcsetpgrp(STDIN_FILENO, shell_pgid);
}

int main(){
    if(isatty(STDIN_FILENO)) setup_shell();
    ignore_job_signals();
    string line;
    while(true){
        reap_children();
        char cwd[4096];
        if(!getcwd(cwd,sizeof(cwd))){
            perror("getcwd");
            cwd[0]='\0';
        }
        cout<<"mini:"<<cwd<<"$ "<<flush;
        if(!getline(cin,line)){
            cout<<"\n";
            break;
```



```cpp
            }
            if(line.size()==0) continue;
            Pipeline pl=parse_line(line);
            if(pl.cmds.empty()) continue;
            if(is_builtin(pl.cmds[0].argv[0]) && pl.cmds.size
    ()==1 && !pl.r.in && !pl.r.out && !pl.r.err){
                run_builtin(pl.cmds[0].argv);
                continue;
            }
            exec_pipeline(pl);
        }
        return 0;
    }
```

Output:



```
raj180@Ubuntu:~/Desktop/Wipro$ ^C
raj180@Ubuntu:~/Desktop/Wipro$ g++ -std=c++17 -O2 -Wall Assignment2.cpp -o Assignment2
raj180@Ubuntu:~/Desktop/Wipro$ ./Assignment2
mini:/home/raj180/Desktop/Wipro$ sleep 30

^Zmini:/home/raj180/Desktop/Wipro$ jobs
[1] 50933 running       sleep 30
mini:/home/raj180/Desktop/Wipro$ sudo whoami
[sudo] password for raj180:
root
mini:/home/raj180/Desktop/Wipro$ ls
Assignment1      Assignment2      Assignment3      backup.sh    suite.sh
Assignment1.cpp  Assignment2.cpp  Assignment3.cpp  logwatch.sh  update_cleanup.sh
mini:/home/raj180/Desktop/Wipro$ cd c..
cd: No such file or directory
mini:/home/raj180/Desktop/Wipro$ cd ..
mini:/home/raj180/Desktop$ ls
project  Wipro
mini:/home/raj180/Desktop$ cd Wipro
mini:/home/raj180/Desktop/Wipro$ ls
Assignment1      Assignment2      Assignment3      backup.sh    suite.sh
Assignment1.cpp  Assignment2.cpp  Assignment3.cpp  logwatch.sh  update_cleanup.sh
mini:/home/raj180/Desktop/Wipro$ exit
bye
raj180@Ubuntu:~/Desktop/Wipro$
```