[Lv4]minishell (2/3)

♡ 9

syamashi
2021年3月21日 04:13

前: minishell(1/3) 入力受付とsignal設定

------実装２: 実行コマンド------

void minishell()に、launchフェーズを加えます。

```
void sig_input();
void sig_ign();
int launch(char *line, char **envp);

void minishell(char **envp)
{
   char *line;
   int  ret;

   ret = 0;
   while (1)
   {
        ft_putstr_fd(MINISHELL, STDERR);
        sig_input();
        if ((get_next_line(0, &line) == 0))
        {
            ft_putstr_fd("exit\n", STDERR);
            exit (ret);
        }
        sig_ign();
        // 実行
        ret = launch(line, envp);
        free(line);
    }
}

void sig_input()
{
    if (signal(SIGINT, sig_int_input) == SIG_ERR)
    {
        ft_putstr_fd(strerror(errno), STDERR);
        exit(1);
    }
    if (signal(SIGQUIT, sig_quit_input) == SIG_ERR)
```

```
        {
            ft_putstr_fd(strerror(errno), STDERR);
            exit (1);
        }

    }

    void sig_ign()
    {
        if (signal(SIGINT, SIG_IGN) == SIG_ERR)
        {
            ft_putstr_fd(strerror(errno), STDERR);
            exit(1);
        }
        if (signal(SIGQUIT, SIG_IGN == SIG_ERR))
        {
            ft_putstr_fd(strerror(errno), STDERR);
            exit (1);
        }
    }




    int launch(char *line, char **envp)
    {
        char *argv[] = {NULL, NULL};
        int pid;
        int status;
        char *tmp;
        char *path;

        argv[0] = line;
        pid = fork();
        // forkエラー処理。
        if (pid < 0)
        {
            ft_putstr_fd(strerror(errno), 2);
            exit(1);
        }
        // 子プロセスの処理
        if ((pid == 0)
        {
            errno = 0;
            path = ft_strjoin("/bin/", line);
            // 実行できるbinallyがpathに存在すればexecveはEXITする
            execve(path, argv, envp);
            // 実行できるbinallyが存在しなかった場合
            if (errno)
            {
                ft_putstr_fd(strerror(errno), 2);
                ft_putstr_fd("\n", 2);
                exit(errno);
            }
        }
        // 親プロセスの処理
```

```
        if (waitpid(pid, &status, 0) < 0)
        {
            ft_putstr_fd(strerror(errno), 2);
            ft_putstr_fd("\n", 2);
            exit(errno);
        }
        return (WEXITSTATUS(status));
    }
```

launch()のゴールは、execveを呼んで、コマンドを実行させることです。

・execve(char *path, char **argv, char **envp)

https://linuxjm.osdn.jp/html/LDP_man-pages/man2/execve.2.html

> 成功すると execve() は返らない。エラーの場合は -1 を返し、 errno を適切に設定する。

Q. 成功すると返らないとは？

A. execveでEXITする、ということ。

Q. 成功とは？

A. 絶対パスに実行可能なbinallyがあると成功します。

サンプルコードは、path = "/bin/<line>"で作成しています。

たとえば/bin/lsは成功します。

Q. 実行可能なbinallyとは？

A. /bin/には、ワカモレ標準搭載のコマンドが設置されています。ファイル形式がバイナリです。普段使っているechoやlsコマンドは、ここにある実行ファイルが働いています。

Q. 失敗する場合とは？

A. /bin/aaaa は存在しないので、失敗します。

chmod 000 /bin/echo

とすれば、/bin/echoがpermission deniedで怒られます。

いずれもexecveではEXITせず、その先のerrno処理まで進みます。

Q. exitしたら、minishellは終わってしまう？

A. fork()しているので、exitするのはminishellではありません。fork()をした時点で、minishellの分身タスク（子プロセス）が生まれます。本体のminishellを親プロセスと呼びます。

・int fork()

https://linuxjm.osdn.jp/html/LDP_man-pages/man2/fork.2.html

子プロセスのID(pid)を戻り値で返します。

fork()を唱えた時点で、コードが分裂し、親と子が並列処理を始めます。

子プロセスはpid==0を得ます。

子プロセスに働かせるため、pid == 0の分岐でexecveを呼びましょう。

親プロセス(minishell本体)は、子プロセスのpid>0を得ます。

親は子プロセスの仕事が終わるまでwaitpid()で待機します。

・wait()

https://linuxjm.osdn.jp/html/LDP_man-pages/man2/wait.2.html

目滑りしますが、以下の4マクロは何かに使えそうです。

```
・WIFEXITED(status)
子プロセスが正常に終了した場合に真を返す。 「正常に」とは、 exit(3) か _exit(2) が呼び出された

・WEXITSTATUS(status)
子プロセスの終了ステータスを返す。 終了ステータスは status 引き数の下位 8ビットで構成されており、

・WIFSIGNALED(status)
子プロセスがシグナルにより終了した場合に真を返す。

・WTERMSIG(status)
子プロセス終了の原因となったシグナルの番号を返す。 このマクロを使用するのは WIFSIGNALED が真を返
```

・課題にあるbuiltinコマンドとは？組み込み関数とは？

「echoをbuiltinでimplementする」ですが、ワカモレ搭載の/bin/echoをexecveで呼ばず、自作echo関数で実現してほしい、ということです。

関数を作るのであって、binallyを作るわけではありません。

なのでexecveに入りません。終わり方も、exitではなくreturnです。

子プロセスも必要ないので、forkに入る前にbuiltinコマンドか判定し、処理を分岐させます。

```
bool builtin_table(char *line);
int builtin_function(char **argv, char **envp);

int launch(char *line, char **envp)
{
    char *argv[] = {NULL, NULL};
    int pid;
    int status;
    char *tmp;
    char *path;

    argv[0] = line;

    // builtinコマンドか判定
    if (builtin_table(line))
       return (builtin_function(argv, envp))

    pid = fork();
    ...
}

bool builtin_table(char *line)
{
    if (!ft_strncmp(line, "echo", 5))
        return (true);
    if (!ft_strncmp(line, "exit", 5))
        return (true);
    return (false);
}

int builtin_function(char **argv, char **envp)
{
    if (!ft_strncmp(argv[0], "echo", 5))
        return (msh_echo());
    if (!ft_strncmp(argv[0], "exit", 5))
        return (msh_exit());
}
```

・pathの話

例ではexecveのpathを、/bin/<line>と決め打ちして投げています。

straceで動作を確認すると、$PATHに記載されているパスをすべて探索する挙動をしています。

```
bash-3.2$ echo $PATH
/Users/syamashi/homebrew/bin:/Users/syamashi/.nodebrew/current/bin:/Users/syamash
rs/syamashi/homebrew/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/

# lsとコマンドを投げると
# hitしない
/Users/syamashi/homebrew/bin/ls
# hitしない
/Users/syamashi/.nodebrew/current/bin/ls
# hitしない
/Users/syamashi/.brew/bin:/Users/syamashi/homebrew/bin/ls
# hitしない
/usr/local/bin/ls
# hitしたので、execveに入ってexit
/bin/ls
```

コロン単位でパスを区切って、コマンドを末尾に付与して、探索します。

・もしunset PATHをすると？

unsetは環境変数の削除コマンドです。

PATHがなくなると/binにいきつけなくなるので、ワカモレ搭載コマンドが動かなくなります。

```
# PATHを削除
bash-3.2$ unset PATH

# lsが失敗する
bash-3.2$ ls
bash: ls: No such file or directory

# 絶対パスなら、実行ファイルが呼び出せます
bash-3.2$ /bin/ls
files
```

Q. builtin関数も動かなくなる？
A. pathに関係ないので動きます。コードを見ても、execveへの分岐に入らず、builtin_tableに入って処理されています。

ここまでで、引数を必要としないワカモレコマンド(ls, date, export, env)と、builtin_tableへの橋渡しができました。

execveで必要なのは、char **argvとchar **envpだけです。
なので、実行担当の人は、これらが来ると想定して、pipeやリダイレクト、引数受け取りなど、オプション機能を少しずつ拡張していけると思います。

・多段パイプの実装チュートリアル。
https://keiorogiken.wordpress.com/2017/12/15/%E3%82%B7%E3%82%A7%E3%83%AB%E3%81%AE%E5%A4%9A%E6%AE%B5%E3%83%91%E3%82%A4%E3%83%97%E3%82%92%E8%87%AA%E4%BD%9C%E3%81%97%E3%81%A6%E3%81%BF%E3%82%8B/

・builtin関数で使える便利な関数3つ。
1. cd
- chdir: pathを投げるだけで移動してくれる、すごいやつ。
- getcwd: 今いる場所を表示してくれる。今いるディレクトリを消されたときに、パスの取り直しで使えます。
2. pwd
- getcwdではなく、$PWDから取得していそうですが、like bashなのでgetcwdでもいい気がします。試しにファイル名を大文字指定で移動し、pwdしてみてください。しなくてもいいけど。
3. その他

- stat

execveに失敗した場合、さらにerrnoの分岐があります。

statは、ファイルがディレクトリなのか、どんな権限があるのか、ファイルの状態を受け取るコマンドです。statによって、出力エラーメッセージが

permission deniedとか。

is a directoryとか。

になります。こだわる場合、判定に役立ちます。

-> 次：minishell(3/3) : 入力文字の解析