# Première interrogation écrite de PDS (Groupe 6 - 2021/2022)

Prénom :

NOM :

## Question 1 :

Expliquez pourquoi `lseek` ou `pread` échouent lorsque le descripteur de fichier passé en paramètre concerne un tube ou une socket par exemple.

## Question 2 :

On considère que le programme suivant est compilé et disponible dans votre répertoire courant sous le nom `./env_mod` :

```
#include <stdio.h>
#include <stdlib.h>

void modify_env() {
    char *env_var_str = getenv("ENV_VAR");
    env_var_str[0] = "A";
}

void print_env() {
    char *env_var_str = getenv("ENV_VAR");
    printf("%s\n", env_var_str);
}

int main(void) {
    modify_env();
    print_env();
    return 0;
}
```

Après exécution des deux premières commandes ci dessous, qu’afficheront les deux commandes suivantes ?
(Renseignez vos réponses directement sous les commandes).

```
$ export ENV_VAR=1head
$ echo $ENV_VAR
> 1head

$ ./env_mod
>


$ echo $ENV_VAR
>
```

## Question 3 :

On souhaite récupérer le nom d’utilisateur (username) associé à un identifiant utilisateur (user ID) passé en paramètre. Implémentez le reste du programme en partant du code donné et en vous servant de la page de manuel fournie en annexe.

Veillez (comme à chaque fois) à produire du code **ROBUSTE** et **EXPLICITE**.

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {
    if (argc != 2) {
        fprintf(stderr, "Wrong number of args\nUsage : %s uid\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    char *endptr;
    errno = 0;
    unsigned long parsed_uid = strtoul(argv[1], &endptr, 10);

    if (errno != 0) {
        perror("strtoul");
        exit(EXIT_FAILURE);
    }

    if (endptr == argv[1]) {
        fprintf(stderr, "Error : Could not parse uid from %s\n", argv[1]);
        exit(EXIT_FAILURE);
    }

    uid_t uid = parsed_uid;
















}
```

## NAME

getpwuid - get password file entry

## SYNOPSIS

```
#include <sys/types.h>
#include <pwd.h>

struct passwd *getpwuid(uid_t uid);
```

## DESCRIPTION

The getpwuid() function returns a pointer to a structure containing the broken-out fields of the record in the password database that matches the user ID uid.

The passwd structure is defined in <pwd.h> as follows:

```
    struct passwd {
        char   *pw_name;       /* username */
        char   *pw_passwd;     /* user password */
        uid_t   pw_uid;        /* user ID */
        gid_t   pw_gid;        /* group ID */
        char   *pw_gecos;      /* user information */
        char   *pw_dir;        /* home directory */
        char   *pw_shell;      /* shell program */
    };
```

## RETURN VALUE

The getpwuid() function returns a pointer to a passwd structure, or NULL if the matching entry is not found or an error occurs. If an error occurs, errno is set to indicate the error. If one wants to check errno after the call, it should be set to zero before the call.

The return value may point to a static area, and may be overwritten by subsequent calls to getpwuid(). (Do not pass the returned pointer to free(3).)

## ERRORS

0 or ENOENT or ESRCH or EBADF or EPERM or ...
    The given name or uid was not found.

EINTR  A signal was caught; see signal(7).

EIO    I/O error.

EMFILE The per-process limit on the number of open file descriptors has been reached.

ENFILE The system-wide limit on the total number of open files has been reached.

ENOMEM Insufficient memory to allocate passwd structure.

ERANGE Insufficient buffer space supplied.

## FILES

/etc/passwd
    local password database file

## ATTRIBUTES

For an explanation of the terms used in this section, see attributes(7).

| Interface | Attribute | Value |
|---|---|---|
| getpwuid() | Thread safety | MT-Unsafe race:pwuid locale |

## CONFORMING TO

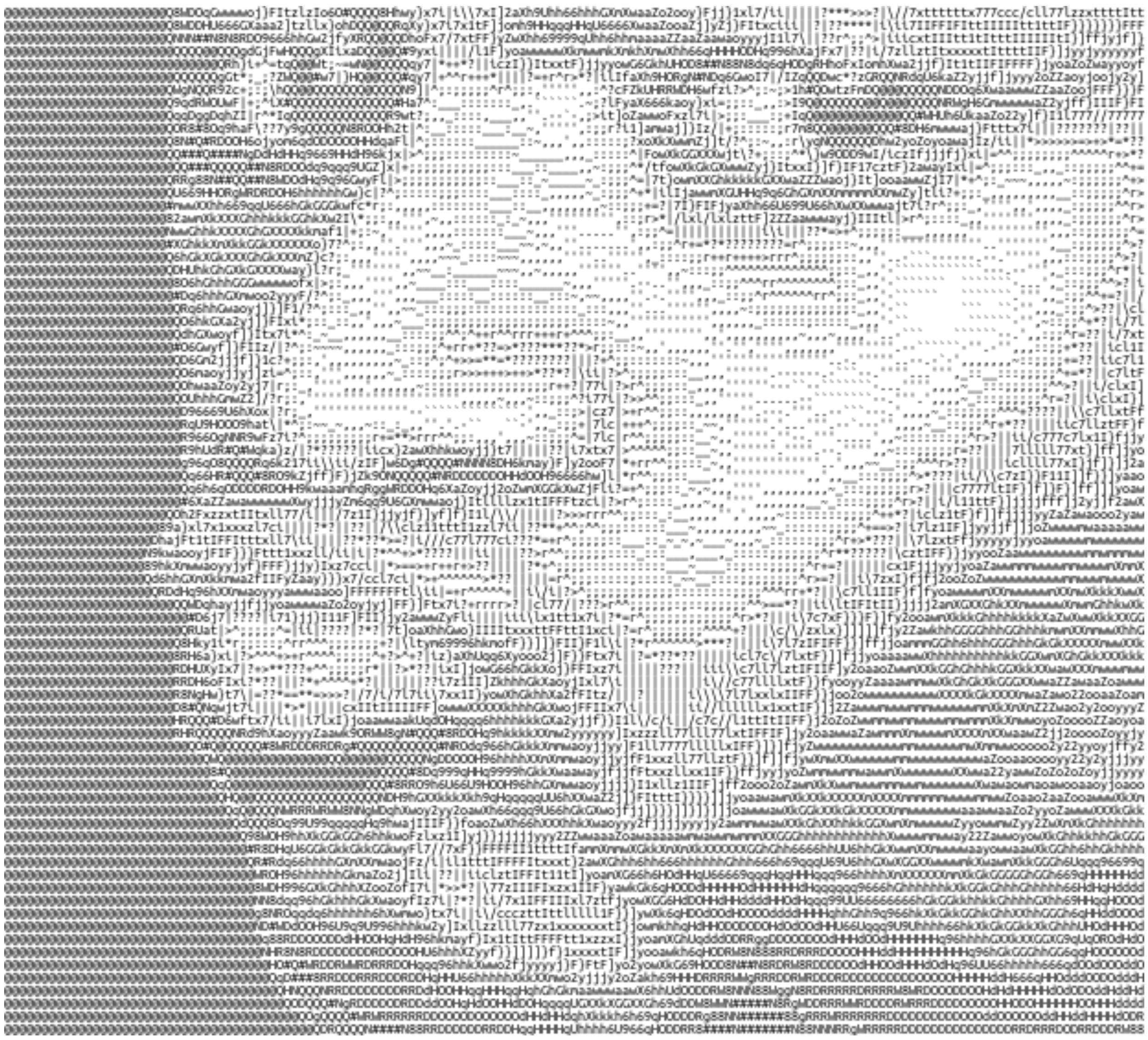
POSIX.1-2001, POSIX.1-2008, SVr4, 4.3BSD.  The pw_gecos field is not specified in POSIX, but is present on most implementations.

## NOTES

The formulation given above under "RETURN VALUE" is from POSIX.1-2001. It does not call "not found" an error, and hence does not specify what value errno might have in this situation. But that makes it impossible to recognize errors. One might argue that according to POSIX errno should be left unchanged if an entry is not found. Experiments on various UNIX-like systems show that lots of different values occur in this situation: 0, ENOENT, EBADF, ESRCH, EWOULDBLOCK, EPERM, and probably others.

The pw_dir field contains the name of the initial working directory of the user. Login programs use the value of this field to initialize the HOME environment variable for the login shell. An application that wants to determine its user's home directory should inspect the value of HOME (rather than the value getpwuid(getuid())->pw_dir) since this allows the user to modify their notion of "the home directory" during a login session.

## EXAMPLES



## SEE ALSO

endpwent(3), fgetpwent(3), getgrnam(3), getpw(3), getpwent(3), getspnam(3), putpwent(3), setpwent(3), passwd(5),

## COLOPHON

This page is part of release 5.13 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.