



#include <dirent.h>

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <string>

#include <memory>

#include <vector>

#include <stdlib.h>

#include <malloc.h>

#include <sys/stat.h>

#include <sys/wait.h>

#include <limits.h>

#include <errno.h>

#include <pthread.h>

#include <sys/mman.h>

#define PATH "/home/consta/practice1/1/Test1/"

#define BUFFER\_SIZE 256

#include <stdio.h>

#include <stdlib.h>

#include <sys/mman.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

void\* create\_shared\_memory(size\_t size)

{

// Our memory buffer will be readable and writable:

int protection = PROT\_READ | PROT\_WRITE;

// The buffer will be shared (meaning other processes can access it), but

// anonymous (meaning third-party processes cannot obtain an address for it),

// so only this process and its children will be able to use it:

int visibility = MAP\_SHARED | MAP\_ANONYMOUS;

// The remaining parameters to `mmap()` are not important for this use case,

// but the manpage for `mmap` explains their purpose.

return mmap(NULL, size, protection, visibility, -1, 0);

}

bool is\_pseudodirectory(struct dirent\* ent)

{

return strcmp(ent->d\_name, (".")) == 0 || strcmp(ent->d\_name, ("..")) == 0;

}

void get\_old\_name(char\* path, struct dirent\* ent, char\* old\_file\_name)

{

sprintf(old\_file\_name, "%s%s", path, ent->d\_name);

}

void get\_new\_path(char\* path, struct dirent\* ent, char\* new\_path)

{

sprintf(new\_path, "%s%s%s", path, ent->d\_name, "/");

}

bool is\_directory(char\* path\_, struct stat\* st)

{

stat(path\_, st);

return S\_ISDIR(st->st\_mode);

}

void UpdateMin(int\* size1, int size2, char\* path1, char\* path2)

{

if (\*size1 <= size2)

return;

\*size1 = size2;

strcpy(path1, path2);

}

int\* min\_count\_file = (int\*)create\_shared\_memory(4);

char\* dir\_name = (char\*)create\_shared\_memory(BUFFER\_SIZE);

// nya

void search\_file(char\* path)

{

DIR\* dir = opendir(path);

int loc\_count\_file = 0;

while (struct dirent\* ent = readdir(dir))

{

struct stat sb;

char file\_name[BUFFER\_SIZE] = "";

get\_old\_name(path, ent, file\_name);

if (stat(file\_name, &sb) == -1)

{

continue;

}

if (is\_pseudodirectory(ent))

{

continue;

}

if (!(sb.st\_mode & S\_IFREG))

{

continue;

}

loc\_count\_file++;

}

closedir(dir);

// printf("%d\n", loc\_count\_file);

UpdateMin(min\_count\_file, loc\_count\_file, dir\_name, path);

}

char\* rec\_dir(char\* path) // DIRECTORY\_NAME

{

// printf("%s", "odin");

search\_file(path);

DIR\* dir = opendir(path);

std::vector<std::unique\_ptr<char[]>> paths;

std::vector<int> procs;

while (struct dirent\* ent = readdir(dir))

{

struct stat sb;

// printf("%s", "dva");

char dir\_path[BUFFER\_SIZE] = "";

get\_old\_name(path, ent, dir\_path);

if (!(is\_directory(dir\_path, &sb)))

{

continue;

}

// printf("%s", "tri");

if (is\_pseudodirectory(ent))

{

continue;

}

// printf("%s", "chert");

paths.emplace\_back(new char[BUFFER\_SIZE]());

get\_new\_path(path, ent, paths.back().get());

int pid = fork();

switch (pid)

{

case -1:

printf("Process error\n");

exit(1);

case 0:

rec\_dir(paths.back().get());

exit(0);

default:

procs.push\_back(pid);

break;

}

}

for (int i = 0; i < procs.size(); i += 1)

{

int status;

waitpid(procs[i], &status, 0);

}

closedir(dir);

return dir\_name;

}

int main(int argc, char\* argv[])

{

\*min\_count\_file = INT\_MAX;

printf("%s\n", rec\_dir(PATH));

return 0;

}