Operating Systems hw2 by Zhaoxi Wu

OS HW2 by Zhaoxi Wu

The program consists three files:

1. main.c
2. mynum.h
3. mystring.h

"mynum.h" constructs a chain to store numbers. For example,

```
struct mynum{
  int num;
  struct mynum *next;
};
```

"mystring.h" constructs a queue to store commands. For example,

```
struct mystring{
  char str[20];
  struct mystring *next;
};
```

"main.c" consists five chains to make a hash table and one queue to store commands. For example,

```
// Create five buckets for the hash table
struct mynum *head01 = create_num();
struct mynum *head23 = create_num();
struct mynum *head45 = create_num();
struct mynum *head67 = create_num();
struct mynum *head89 = create_num();
```

- When a number x satisfies $x \bmod 10 = 0 \text{ or } 1$, then x will be put into chain 1, namely head01.
- When a number x satisfies $x \bmod 10 = 2 \text{ or } 3$, then x will be put into chain 2, namely head23.
- When a number x satisfies $x \bmod 10 = 4 \text{ or } 5$, then x will be put into chain 3, namely head45.
- When a number x satisfies $x \bmod 10 = 6 \text{ or } 7$, then x will be put into chain 4, namely head67.
- When a number x satisfies $x \bmod 10 = 8 \text{ or } 9$, then x will be put into chain 5, namely head89.

Generate threads:

```
// Create threads
pthread_t threads[command_number];
memset(&threads, 0, sizeof(threads));
pthread_mutex_init(&mut, NULL);
```

Use threads:

```
// Create threads to do the tasks
pthread_create(&threads[current_thread_num], NULL, thread1, (void *) NULL);
// Wait for threads to finish
pthread_join(threads[current_thread_num], NULL);
current_thread_num++;
```

Child process receives commands from user

```
close(*read_fd);
// Child process gets input
while(1)
{
  // If we use gets, we will get warnings. We use fgets, but the end of the string will be \n
  fgets(readbuffer, 20, stdin);
  printf("the input are: %s", readbuffer);
  // Child writes, parent reads
  write(*write_fd, readbuffer, sizeof(readbuffer));
  // Stop if meets exec
  if(!strcmp("exec\n", readbuffer))
  {
    break;
  }
  // Clear the buffer
  strcpy(readbuffer, "");
}
close(*write_fd);
```

Parent process stores those commands

```
close(*write_fd);
// Create a chain to keep commands
struct mystring *command = create_string();
// Trace the number of user-input commands
int command_number = 0;
while(1)
{
  nbytes=read(*read_fd, readbuffer, sizeof(readbuffer));
  printf("the parent receive %d bytes data: %s\n", nbytes, readbuffer);
  // Stop if meets exec
  if(!strcmp("exec\n", readbuffer))
  {
    break;
  }
  // Keep all these commands
  add_string(command, readbuffer);
  command_number ++;
  // Clean the buffer
  strcpy(readbuffer, "");
}
close(*read_fd);
```

Parent process parses those commands and generates threads

```
// Trace the number of current threads
int current_thread_num = 0;
while(temp_command->next != NULL)
{
  printf("The string is %s\n", temp_command->next->str);
  // Parse strings
  char delims[] = " ";
  char *res = NULL;
  res = strtok(temp_command->next->str, delims);
  res = strtok(NULL, delims);
  temp_int = atoi(res);
  temp_com = temp_command->next->str[0]-'a';
  // Create threads to do the tasks
  pthread_create(&threads[current_thread_num], NULL, thread1, (void *) NULL);
  // Wait for threads to finish
  pthread_join(threads[current_thread_num], NULL);
  current_thread_num++;
  // Next command
  temp_command = temp_command->next;
}
```

Compile the file by the following command in the shell:

```
g++ -pthread -o a.out main.c
```

- Use the command "gcc" to compile files would get the error "initializer element is not constant", thus I use "g++".
- In order to use threads, I use "-pthread" in the command.
- The executable file is called "a.out".

Run the program by the following command in the shell:

```
./a.out
```

The test is the following:

1. User inputs somes commands to the child process.
2. Child process sends all these commands to parent process by pipe until it meets "exec".
3. parent process generates a hash table and executes those received commands by constructing some threads.

There is an example:
User inputs:

```
add 1
add 2
add 3
add 4
```

```
search 1
delete 1
search 1
```

There are totally four number in the hash table. The first time we search for 1, the answer will be true or 1. The second time we search for 1, the answer will be false or -1.

The outputs:

```
add 1
the input are: add 1
the parent receive 20 bytes data: add 1

add 2
the input are: add 2
the parent receive 20 bytes data: add 2

add 3
the input are: add 3
the parent receive 20 bytes data: add 3

add 4
the input are: add 4
the parent receive 20 bytes data: add 4

search 1
the input are: search 1
the parent receive 20 bytes data: search 1

delete 1
the input are: delete 1
the parent receive 20 bytes data: delete 1

search 1
the input are: search 1
the parent receive 20 bytes data: search 1

exec
the inpput are: exec
the parent receive 20 bytes data: exec

The string is add 1

result: add 1
The string is add 2

result: add 2
The string is add 3

result: add 3
The string is add 4

result: add 4
The string is search 1

result: search 1
1
```

```
The string is delete 1

result: delete 1
The string is search 1

result: search 1
-1
--bucket 1---
--bucket 2---
The number is 3
The number is 2
--bucket 3---
The number is 4
--bucket 4---
--bucket 5---
```