CS2002 170025298 Tutor: Maywan Fayed
Date: 14th April 2019
Practical 5

Part 1:
Design & implementation:
The code has logic as follows:
1. Since the input is from the command line, the first step is to link all arguments to one single sentence by using link_args()
2. The task is make a chain of 10 processes, so by calling create_process(), it will create processes by using fork() and pass sentence as argument.
3. In the create_process(), we have to judge whether current chain is less than 10, if so, the function will switch two letters and create a new process by using fork()
4. Since current number of chain cannot pass to different process, so the function will create two pipe and write (current depth + 1) to the first pipe and write the changed sentence to the second pipe.
5. The child processes will get the value from pipe and call create_process() again. Values from pipes will be used to finish to finish step 3.
6. If the depth is bugger than 10, the loop will stop.
7. Since the parent process may close after the program finished, so I added wait(NULL) to parent in order to prevent the condition that getppid() will return as 1 or some strange number.

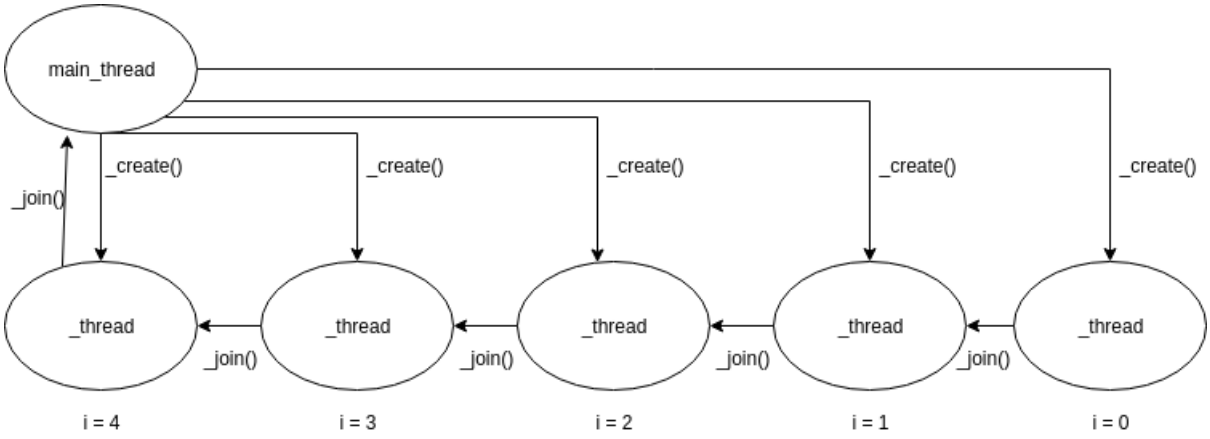
Difficulty:
Since there are one thing that confused me:
From the question sheet, switch two words will not switch blank space, so I added the condition that indices will not choose blank space. Since to prevent misunderstanding, I threw a better choice of switching: Ignoring linking words, just switch two random integer i1 i2 in orger to choose arg[i1], arg[i2], then choose another two random integet i3 and i4 and switch words in arg[i1][i3], arg[i2] [i4] and print sentence by using while loop.

Testing:

```
hl74@pc5-012-l:~/Documents/CS2002/W12Practical/rumours $ ./rumours I am good!
pid: 21940 Swapped indices 10, 8
New process: 21941, parent: 21940
pid: 21941 received string:  I am go!do
pid: 21941 Swapped indices 10, 8
New process: 21942, parent: 21941
pid: 21942 received string:  I am good!
pid: 21942 Swapped indices 9, 1
New process: 21943, parent: 21942
pid: 21943 received string:  d am gooI!
pid: 21943 Swapped indices 1, 4
New process: 21944, parent: 21943
pid: 21944 received string:  m ad gooI!
pid: 21944 Swapped indices 6, 1
New process: 21945, parent: 21944
pid: 21945 received string:  g ad mooI!
pid: 21945 Swapped indices 6, 10
New process: 21946, parent: 21945
pid: 21946 received string:  g ad !ooIm
pid: 21946 Swapped indices 9, 6
New process: 21947, parent: 21946
pid: 21947 received string:  g ad Ioo!m
pid: 21947 Swapped indices 8, 4
New process: 21948, parent: 21947
pid: 21948 received string:  g ao Iod!m
pid: 21948 Swapped indices 4, 9
New process: 21949, parent: 21948
pid: 21949 received string:  g a! Iodom
pid: 21949 Swapped indices 1, 9
New process: 21950, parent: 21949
pid: 21950 received string:  o a! Iodgm
```

Since all results shows exactly like what problem sheets shows. There exactly no trouble happens to this part.

Part 2:
The thread join diagram shows as follows:



1. Describe:
This program will do the following steps:
i> Create the first thread and take arguments do following things:
&thread is the thread that used to store the thread id.
Worker is the starting function of the thread
(void*) thread is the value of thread since in first step this value is null.
ii> using for loop and call pthread_create() again and take first created thread as the last argument and store the address of this thread as thread.
Iii> Loop until i = 4, create the last thread and take thread when I = 3 as last argument, then join the thread when I = 4.

Assuming pthread_create() has return value thread, then this function can be wrote as a fake code like this:
pthread_create(pthread_create(pthread_create(pthread_create(pthread_create()))))
which is like a list.

So in the worker the data is a thread including arguments(&data, NULL, worker, (void*)thread) for which this thread is the previous stage (I = 3) of the thread such that the process shows as graph above.

2. Testing:

Conclusion above can be proved with exactly running this code: since we print the address of each thread and print it from the console, we can get the following result:

```
hl74@pc5-012-l:~/Documents/CS2002/W12Practical/rumours $ ./test
i : 0, addr : 140227195795200----------
i : 1, addr : 140227187402496----------
i : 2, addr : 140227179009792----------
i : 3, addr : 140227170617088----------
i : 4, addr : 140227162224384----------
--------------
first addr : 140227162224384
----------
data: 0-tid: 140227195795200-cont : 0
data: 140227195795200-tid: 140227187402496-cont : 1
data: 140227187402496-tid: 140227179009792-cont : 2
data: 140227170617088-tid: 140227162224384-cont : 3
data: 140227179009792-tid: 140227170617088-cont : 4

output : 5
```

Where we can find that the first address is the address when I = 4 and every data in worker contains the thread id of previous one. This test completely proved my theorem and current tid stores the value of previous one. The detailed test program were in testcase.c and we can use ‘make’ to get the similar output.

Part 3:
Design:
In this part, logic of three parts of the program will be described here:
1. Preparing Part:
i> Since we can only pass one parameter into some threads and the parameter has to be thread count, as a result, some global variables are settled.
(char*) input[3] are array of Strings, which is used to store the input value: User-name, Password Hex and Prefix of the password from the standard input.
(int*) unknown are array of integers, which is used to store the position of the unknown value in prefix.
(int) amount stores the value of unknown values, which shows as ‘.’ in prefix.
ii> Malloc each elements in input[3] and use scanf to get data from standard input.
iii> create the free function and free the string in the end.
iv> create first thread and set thread_start_function() as the start of the thread.

2. Decryption Part:
i> Use strcpy() to copy from input[2] (refers to ‘prefix password’) to str as a backup.
ii> Create a string ‘attempt’ which will be used to attempt all possible solutions to get the correct password.
iii> Initialise ‘attempt’
iv> By calling set_string_position, change the initial value into correct character, after calling increment String and finally insert attempt to the copy of prefixed and compare the result with Hex.
v> return success / failure

3. Thread create/join:
i> Create an array of threads
ii> Calling pthread_create() and using for loop to create threads.
Iii> Use pthread_join() to wait all threads finished working.
iv> Since from the requirement we has to terminate the thread when codes finished, in case we

For the multi password, we need to insert all input into the queue and retrieve them with FIFO laws. Due to the time issue, I discard this extension.

Testing:
For data:
u000000 csJPmyY1CRmJM zliot... zliotnhx 8994
the output is:

```
hl74@pc5-012-l:~/Documents/CS2002/W12Practical/shadowbreaker $ ./shadow_breaker  <pwd_hashes.txt
Start u000000
Result found: zliotnhx
```

which finished the requirement.