Homework 5

Lim Saeyeon (#21102054)

Recursive Fibonacci (Fibonacci_rec.c)

```
CS21102054@nshcdell ~/HW5 (8.324s)
./fibonacci_rec
Fibonacci of 90:
^C
```

```
CS21102054@nshcdell ~/HW5
gdb ./fibonacci_rec
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.2) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
pwndbg: loaded 147 pwndbg commands and 47 shell commands. Type pwndbg [--shell | --all] [filter] for a list.
pwndbg: created $rebase, $ida GDB functions (can be used with print/break)
Reading symbols from ./fibonacci_rec...
(No debugging symbols found in ./fibonacci_rec)
------- tip of the day (disable with set show-tips off) -------
Use Pwndbg's config and theme commands to tune its configuration and theme colors!
pwndbg> break fibonacci
Breakpoint 1 at 0x1169
pwndbg> run
Starting program: /home/CS21102054/HW5/fibonacci_rec
Fibonacci of 90:

Breakpoint 1, 0x0000555555555169 in fibonacci ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
──────────────────────────[ REGISTERS / show-flags off / show-compact-regs off ]──────────────────────────
 RAX  0x0
*RBX  0x555555555240 (__libc_csu_init) ◂— endbr64
 RCX  0x0
 RDX  0x0
 RDI  0x0
*RSI  0x5555555592a0 ◂— 'Fibonacci of 90: \n'
 R8   0x0
*R9   0x12
*R10  0x555555556015 ◂— 0x646c6c25000a203a /* ': \n' */
*R11  0x246
*R12  0x555555555080 (_start) ◂— endbr64
*R13  0x7fffffffe4b0 ◂— 0x1
 R14  0x0
 R15  0x0
*RBP  0x7fffffffe3c0 ◂— 0x0
*RSP  0x7fffffffe3a8 —▸ 0x555555555205 (main+66) ◂— mov rsi, rax
*RIP  0x555555555169 (fibonacci) ◂— endbr64
```

```
──────────────────────────[ DISASM / x86-64 / set emulate on ]──────────────────────────
► 0x555555555169 <fibonacci>       endbr64
  0x55555555516d <fibonacci+4>     push   rbp
  0x55555555516e <fibonacci+5>     mov    rbp, rsp
  0x555555555171 <fibonacci+8>     push   rbx
  0x555555555172 <fibonacci+9>     sub    rsp, 0x18
  0x555555555176 <fibonacci+13>    mov    qword ptr [rbp - 0x18], rdi
  0x55555555517a <fibonacci+17>    cmp    qword ptr [rbp - 0x18], 0
  0x55555555517f <fibonacci+22>    jne    fibonacci+31                <fibonacci+31>

  0x555555555181 <fibonacci+24>    mov    eax, 0
  0x555555555186 <fibonacci+29>    jmp    fibonacci+83                <fibonacci+83>
   ↓
  0x5555555551bc <fibonacci+83>    add    rsp, 0x18
──────────────────────────[ STACK ]──────────────────────────
00:0000│ rsp 0x7fffffffe3a8 —▸ 0x555555555205 (main+66) ◂— mov rsi, rax
01:0008│     0x7fffffffe3b0 ◂— 0x0
02:0010│     0x7fffffffe3b8 ◂— 0x5a /* 'Z' */
03:0018│ rbp 0x7fffffffe3c0 ◂— 0x0
04:0020│     0x7fffffffe3c8 —▸ 0x7ffff7de6083 (__libc_start_main+243) ◂— mov edi, eax
05:0028│     0x7fffffffe3d0 —▸ 0x7ffff7ffc620 (_rtld_global_ro) ◂— 0x504ff00000000
06:0030│     0x7fffffffe3d8 —▸ 0x7fffffffe4b8 —▸ 0x7fffffffe708 ◂— '/home/CS21102054/HW5/fibonacci_rec'
07:0038│     0x7fffffffe3e0 ◂— 0x100000000
──────────────────────────[ BACKTRACE ]──────────────────────────
```

```
──────────────────────────────────────────[ BACKTRACE ]──────────────────────────────────────────
 ► 0   0x555555555169 fibonacci
   1   0x555555555205 main+66
   2   0x7ffff7de6083 __libc_start_main+243
──────────────────────────────────────────────────────────────────────────────────────────────────
pwndbg> bt
#0  0x0000555555555169 in fibonacci ()
#1  0x0000555555555205 in main ()
#2  0x00007ffff7de6083 in __libc_start_main (main=0x5555555551c3 <main>, argc=1, argv=0x7fffffffe4b8, init=<optimi
zed out>, fini=<optimized out>, rtld_fini=<optimized out>, stack_end=0x7fffffffe4a8) at ../csu/libc-start.c:308
#3  0x00005555555550ae in _start ()
pwndbg> x/5i $rip
=> 0x555555555169 <fibonacci>:  endbr64
   0x55555555516d <fibonacci+4>:        push   rbp
   0x55555555516e <fibonacci+5>:        mov    rbp,rsp
   0x555555555171 <fibonacci+8>:        push   rbx
   0x555555555172 <fibonacci+9>:        sub    rsp,0x18
pwndbg> x/20xg $rsp
0x7fffffffe3a8: 0x0000555555555205      0x0000000000000000
0x7fffffffe3b8: 0x000000000000005a      0x0000000000000000
0x7fffffffe3c8: 0x00007ffff7de6083      0x00007ffff7ffc620
0x7fffffffe3d8: 0x00007fffffffe4b8      0x0000000100000000
0x7fffffffe3e8: 0x00005555555551c3      0x0000555555555240
0x7fffffffe3f8: 0xcd474868268eba41      0x0000555555555080
0x7fffffffe408: 0x00007fffffffe4b0      0x0000000000000000
0x7fffffffe418: 0x0000000000000000      0x32b8b797e12eba41
0x7fffffffe428: 0x32b8a7d4e6e0ba41      0x0000000000000000
0x7fffffffe438: 0x0000000000000000      0x0000000000000000
pwndbg> info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000555555555169 <fibonacci>
        breakpoint already hit 1 time
pwndbg> info r
rax            0x0                 0
rbx            0x555555555240      93824992236096
rcx            0x0                 0
rdx            0x0                 0
rsi            0x5555555592a0      93824992252576
rdi            0x0                 0
rbp            0x7fffffffe3c0      0x7fffffffe3c0
rsp            0x7fffffffe3a8      0x7fffffffe3a8
r8             0x0                 0
r9             0x12                18
r10            0x555555556015      93824992239637
r11            0x246               582
r12            0x555555555080      93824992235648
r13            0x7fffffffe4b0      140737488348336
r14            0x0                 0
r15            0x0                 0
rip            0x555555555169      0x555555555169 <fibonacci>
eflags         0x297               [ CF PF AF SF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
```

The code is taking a long time to produce the output because the `fibonacci` function is implemented using a recursive approach without memoization. This recursive implementation recalculates the same Fibonacci values multiple times, which leads to an exponential time complexity, specifically (O(2^n)).

For each call to `fibonacci(n)`, the function calls itself for both `fibonacci(n-1)` and `fibonacci(n-2)`. This creates a binary tree of function calls, where each level of recursion generates twice as many calls as the previous level. Thus, calculating Fibonacci values for larger inputs, such as `n=90`, leads to an enormous number of redundant function calls.

Additionally, in the `gdb` debug session, we can observe the deep stack traces, indicating the significant memory and processing overhead due to the high number of recursive calls.

revised Fibonacci (Fibonacci.c)

HW5 > C fibonacci.c

```c
#include <stdio.h>

unsigned long long fibonacci(unsigned long long n)
{
    if (n <= 1) return n;

    unsigned long long prev = 0, curr = 1, next;
    for (unsigned long long i = 2; i <= n; i++)
    {
        next = prev + curr;
        prev = curr;
        curr = next;
    }
    return curr;
}

int main() {
    unsigned long long i;
    unsigned long long n = 90;

    printf("Fibonacci of %lld: \n", n);

    for(i = 0; i<n; i++)
    {
        printf("%lld ", fibonacci(i));
    }
    printf("\n");
    return 0;
}
```

```
CS21102054@nshcdell:~/HW5 (0.019s)
./fibonacci
Fibonacci of 90:
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418 3
17811 514229 832040 1346269 2178309 3524578 5702887 9227465 14930352 24157817 39088169 63245986 102334155 16558014
1 267914296 433494437 701408733 1134903170 1836311903 2971215073 4807526976 7778742049 12586269025 20365011074 329
51280099 53316291173 86267571272 139583862445 225851433717 365435296162 591286729879 956722026041 1548008755920 25
04730781961 4052739537881 6557470319842 10610209857723 17167680177565 27777890035288 44945570212853 72723460248141
 117669030460994 190392490709135 308061521170129 498454011879264 806515533049393 1304969544928657 2111485077978050
 3416454622906707 5527939700884757 8944394323791464 14472334024676221 23416728348467685 37889062373143906 61305790
721611591 99194853094755497 160500643816367088 259695496911122585 420196140727489673 679891637638612258 1100087778
366101931 1779979416004714189
```

This improved code uses an iterative approach to calculate Fibonacci numbers, making it much faster and more efficient. Unlike the recursive version, which has exponential time complexity and can cause stack overflow for large (n), the iterative method has linear time complexity, (O(n)), and uses constant memory. This avoids redundant calculations and reduces execution time significantly.