15-213: Introduction to Computer Systems

Written Assignment #7

This written homework covers Dynamic Memory Allocation and Exceptional Control Flow.

Directions

Complete the question(s) on the following pages with single paragraph answers. These questions are not meant to be particularly long! Once you are done, submit this assignment on Canvas. Below is an example question and answer.

Q: In a few sentences, give two practical reasons why two's complement signed integers are more convenient to work with than one's complement or sign-and-magnitude.

A: Other representations of signed integers (ones-complement and sign-and-magnitude) have two representations of zero (+0 and −0), which makes testing for a zero result more difficult. Also, addition and subtraction of two's complement signed numbers are done exactly the same as addition and subtraction of unsigned numbers (with wraparound on overflow), which means a CPU can use the same hardware and machine instructions for both.

Grading

Each assignment will be graded in two parts:

1. Does this work indicate any effort? (e.g. it's not copied from a homework for another class or from the book)
2. Three peers will provide short, constructive feedback.

This assignment is due on July 28, 2022 by 11:59pm Pittsburgh time (currently UTC−4). Remember to convert this time to the timezone you currently reside in.

Question #1

A. Identify the kind of fragmentation in the below scenarios

Scenario 1: Requested Payload is 8 bytes but the block size is 32 bytes
Scenario 2: Requested Payload is 64 bytes Total Free memory available is 64 bytes but it is not available in one contiguous chunk.

1. Internal fragmentation
2. External fragmentation

B. What is the difference between internal fragmentation and external fragmentation? Is there a way to decrease the amount of internal fragmentation? What about external fragmentation? Answer why or why not for each.

Internal fragmentation is caused by the design of memory blocks, padding/alignment, and explicit policy decisions. It only depends on the pattern of previous requests.

External fragmentation is caused by not having enough contiguous memory for new requests. It depends on the pattern of future requests.

One way to decrease the amount of internal fragmentation is to reduce data structure overhead of memory blocks. Because lower overhead means less extra space used by each memory block, thus less internal fragmentation.

One way to decrease the amount of external fragmentation is to use better policy, such as best-fit. Because it can place each block into better memory position to reduce the chance of external fragmentation.

Question #2

A memory allocator (malloc) has the following properties.

- At the start, there is a prologue and an epilogue (each 8 bytes).
- Each block has 8 byte headers and 8 byte footers.
- The blocks of memory are 16-byte aligned.
- Blocks are coalesced if possible

A. Some statistics for a set of trace files indicate that the best balance between throughput and utilization is achieved when the chunk size is 2048 bytes. How much should the heap expand by when there is no space available for a new request? Try to think about different cases!

Assume the size of the new request is x.
If x is larger than 2048 bytes, we expand the heap by the next multiple of 16 bytes of x.
If x is smaller than 2048 bytes, we expand the heap by 2048 bytes.

B. The below code is executed, draw out how the heap looks after executing every free request.

```
void *block1 = malloc(43);
void *block2 = malloc(5);
free(block1);
void *block3 = malloc(1198);
void *block4 = malloc(515);
void *block5 = malloc(7010);
free(block4);
```

After the first free request:
| prologue | 48 free | 16 alloc | 1984 free | epilogue |

After the second free request:
| prologue | 48 free | 16 alloc | 1200 alloc | 528 free | 7024 alloc | 256 free | epilogue |

Question #3

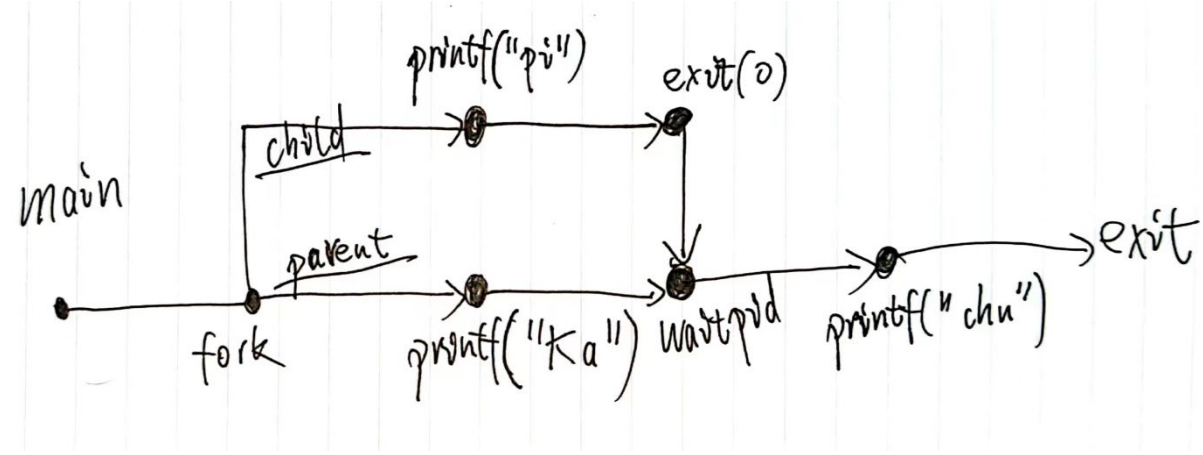
Given the following code snippet:

```
1     int main()
2     {
3        if (fork())
4        {
5           printf("ka");
6           waitpid(-1, NULL, 0);
7        }
8        else
9        {
10          printf("pi");
11          exit(0);
12       }
13       printf("chu");
14       exit(0);
15    }
```

Assume that all system calls succeed and error checking has been omitted.

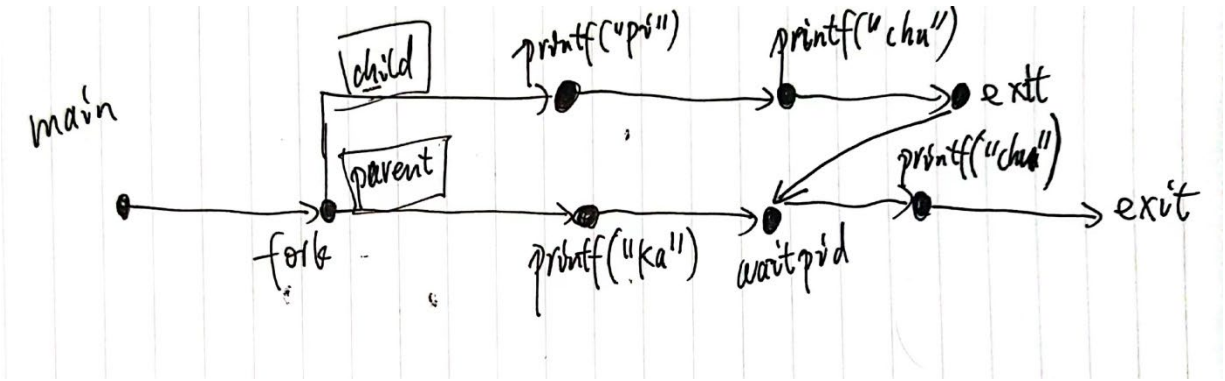
1. Draw the process graph for this program. What are all the possible output sequences?
2. If we remove exit(0); in line 11, now what are all the possible output sequences? Draw the new process graph for this program.

1.



Possible output sequences are:
- pikachu
- kapichu

2.



Possible sequences are:
- pikachuchu
- kapichuchu
- pichukachu