CS342 Operating Systems

Project Assignment 4

Due: December 12, 2015, Saturday. 11:55 am.

Serkan Demirci, 21201619

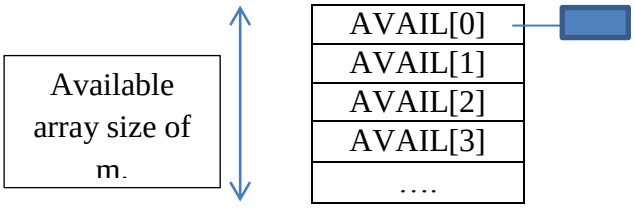
Özgür Öney, 21101821

Part A – Implementation of the project

We have started to work on project by thinking about best and most efficient way to implement. Therefore, we have used some additional study material. Donald E. Knuth's *The Art of Computer Programming* was our main resource used. Among many options in dynamic storage allocation methods, we have chosen the array-based buddy system allocation. This method takes one bit "overhead" in each block, and it requires all blocks to be of length 1, 2, 4, 8 or etc. If a block is not 2^k words long for some integer k, the next higher power of 2 is chosen and extra unused space is allocated accordingly.

The idea of this method is to keep separate lists of available blocks of each size 2^k, where $0 \leq k \leq m$. The entire pool of memory under allocation consists of 2^m words, which we will assume for convenience has the addresses 0 through 2^m -1. Originally, the entire block of 2^m words is available. Later, when a block of 2^k words is desired, and if nothing of this size is available, a larger available block is split into 2 equal parts; ultimately, a block of the right size 2^k will appear. When one block splits into two (each of which is half as large as the original) called buddies. [1]

Such buddy system makes use of a one-bit TAG field in each block, where 0 indicates block is reserved and 1 indicates block is available. In addition to that, blocks in available array also have usual doubly linked list, to ease access and operations. They also have a KVAL value to represent k when their size is 2^k.



Buddy system reservation works as following:

1. Find block
2. Remove from list.
3. Check split required or not.
4. Split

Buddy system liberation, however, works as following:

1. Check whether buddy is available or not.
2. Combine with buddy.
3. Put on list.

Part B – Timing experiments for implemented code

In order to increase the efficiency and get more precise result, we have automated the test. Following lines of code are written in this purpose. In first box, lines that is used to measure the time spent for allocation and deallocation of chunk in specified size of interval is shown and in addition to that, at second box, code to take the average of 100 different results in fixed chunk size in automated way is shown.

To execute the code, two different parameters, chunk size and random number interval should be provided by the way.

```
#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include "buddy.h"

#include <sys/time.h>

#define SPACE 1000

void* allocatedSpace[SPACE+1] = {NULL};

int randomi(int max)

{

   return rand() % (max+1);

}

//size , seed

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

{

   srand(atoi(argv[2]));

   int size = atoi(argv[1]);

            void* chunk = malloc(size*1024);

            int a = binit(chunk,size);

            if (a == -1)

            {

               printf("-1\n");

               return 0;
```

```
        }
        //printf("%d\n",a);
        int i = 0;
        //Calculate time
        struct timeval start, end;
  long mtime, seconds, useconds;
  //START
  gettimeofday(&start, NULL);
  for (i = 0 ; i < 20000; i++)
          allocatedSpace[randomi(SPACE)] = balloc((1 << (randomi(7)+9)) + randomi(128));


        for (i = 0 ; i < 20000 ; i++)
        {
           int r = randomi(SPACE);
           if (allocatedSpace[r] != NULL)
              bfree(allocatedSpace[r]);
           allocatedSpace[r] = NULL;
        }
  //END
  gettimeofday(&end, NULL);
  seconds  = end.tv_sec  - start.tv_sec;
  useconds = end.tv_usec - start.tv_usec;
  mtime = (seconds * 1000000 + useconds);
        printf("%ld\n",mtime);
        return 0;
}
```

```
#!/bin/bash

SEED=$2

SIZE=$1

TIMES=100

SUM=0

for i in $(seq 1 1 $TIMES)

do

   RUN=$(./app $SIZE $SEED)

   #echo "$i : $RUN"

   SUM=$(bc -l <<< "$SUM + $RUN")

done

AVG=$(bc -l <<< "$SUM / $TIMES")

echo $AVG
```

However, table below gives the information about the time spent for allocating and deallocating the fixed size memory in given chunks. Chunk sizes are in the range between 32 KB and 32 MB, as specified in “Project 4 Instructions” on the website.

Chunk Size (bytes)	**Time spent (µ-seconds)**
2^{15}	6508.13
2^{17}	7344.47
2^{19}	7659.53
2^{21}	7875.36
2^{23}	9177.31
2^{25}	17491.84

If we take into account all of these numbers on the table above, it would be appropriate to say that time spent is directly proportional to chunk size. However, as numbers suggest, such proportionality cannot be described as linear but some sort of exponential. This could be because of the system used as well as increasing number of merge/divide operations. In following figure, behavior described above will be demonstrated.

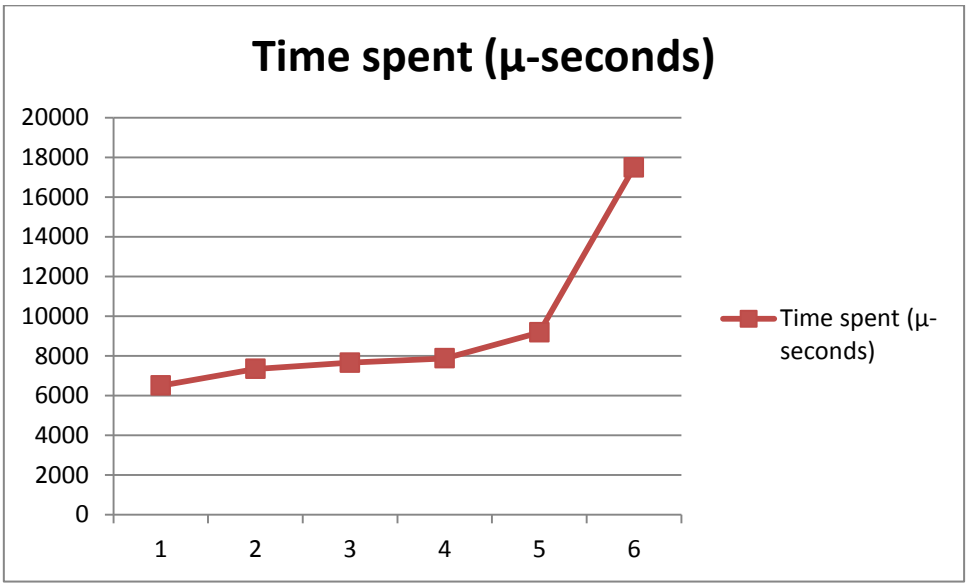


Figure 1: Time spent vs. chunk size

Part C – References

[1] Knuth, D. (1973). *The art of computer programming*. Reading, Mass.: Addison-Wesley Pub. Co., page 442-445.