ALAA ZAKARIYA
BITNINE TASK
2023-AUG

BRIEF

pool.c

Description:

unrestricted storage allocator in its internals calling malloc to obtain huge amount in memory and from it manage distributing this memory between it processes clients and is implemented in first fit algorithm which shall be advantageous over stack for it is randomly not sequentially manageable and quick in performance but still suffer of segmentation of memory which could prevent allocating free required amount for free areas are dispersed among the islands of allocated memory which calls for best fit algorithm rather

Implementation Details

this implementation is similar to one applied in old unix system memory allocator but in a much simplified way. this starts with a free chunk of memory with size of doubles of certain block size i.e. the size determined by number of blocks and the first block of each free area contains a header or data structure contains pointer to next free block in the chain and record of free space that following the header
in blocks units. by that record. i.e the free chunk consists of header and body.header contains a link to next free space and arecord of current free space size and in the initial case the pointer points to the same block and size equal to the whole block.

in case of request the manager loop over free chunks to find
first one that is equal or greater than required amount
and then create new block in the end of free space
and just adjust link and size of the previous as in figure

in case of deallocation the freed amount the chain is searched to make reallocation to adjacent free spaces before and after if there
any and merging them and readjust header link and size.

initial state

header

```
_______________________________________________________
| N 1| S1 |       f r e e s p a c e                     |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

N: pointer to next free space which is pointer to itself
for we have one piece
S: size of following free space

allocating from end

```
_______________________________________________________
| N 1| S1 |                              |N2|S2|xxxxxxx|
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

N1: same link to itself
S1: adjusted by subtracting amount taken by block 2

allocating again

```
_______________________________________________________
| N 1| S1 |               |N3|S3|xxxxxxxx|N2|S2|xxxxxxx|
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

N1: same link to itself
S1: more adjusted by subtracting amount taken by clock 3

block 2 is freed

```
_______________________________________________________
| N 1| S1 |               |N3|S3|xxxxxxxx|N2|S2|       |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

N1: now N1 relocated to point to start of header of block 2
which is next free avilable
S1: more adjusted by subtracting amount taken by clock 3

in case of freeing block 3 instead

```
_______________________________________________________
| N 1| S1 |                             |N2|S2|xxxxxxx|
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

N1: reallocated to link to itself
S1: size is increased by adding the freed amount

and the header is implemented with c struct contains pointer to next variable and size variable and this struct is in union to control block size by any other desirable variable size as desired for implementation

```
union header {
struct {
union header * next;
unsigned size
}s

double align
}
```

and by using header pointer we can jump to free space of any chunk by incrementing the header pointer by one to return in allocation function

more details is on comments with source code

PROS OF THIS IMPLEMENTATION

**** this pool could be extended easily by asking the system to allocate another and linking the new with last pointer in the exited since we are already can handle not adjacent areas

**** compact and simple in data structure and can be handled or interpreted or analyzed by third party process since data structures is embedded into memory itself

**** efficient performance

CONS

**** complexity in allocation and freeing logic

**** though rare to happen but headers in heavy segmentation can reduce size of memory for it is overload and not really used by process

**** processes can scramble the data of other process of another clients so more secure measure should be researched for example by allocate process to hand any request for writing to insure permissions to not changing data

PTEST: pool test bed

testbed tool to test certain condition from options through which you can determine pool size ,chunk sizes and you can make them random. and determine their number. please see the -h help option.

the tool is made by array structs record to hold the data and results of allocating and deallocating every chunks like size and times then the array is looped three times one allocating, 2nd in de allocating, and third in calculating statistics and printing data

NOTE:

if I got leisure of time I would implement a test_bed_map_grapher which not only testing performance but consistency and graphically visualize the memory consumption and free area
and by allocating certain parts with fixed chars like 4444444 or 66666666 then scan the memory byte by byte to assert the integrity and headers and data and free areas

API

```
void *|NULL pool_init(int size)
void *|NULL palloc(int size)
void pfree(void*)
```

in header pool.h