
LUT
University
Operating Systems and System Programming
LAND
OF THE
CURIOUS

Experiment 9 Project4

- Project 4 Simulation of Continuous Dynamic Partition Allocation
- Assume that in the initial state, the available memory space is 640KB, and there is the following request sequence:

- •job1 applies for 130KB
- •job2 applies for 60KB
- •job3 applies for 100KB
- •job2 releases 60KB
- •job4 applies for 200KB

•job3 releases 100KB
•job1 releases 130KB
•job5 applies for 140KB
•job6 applies for 60KB
•job7 applies for 50KB
•job6 releases 60KB

Experiment 9 Project4

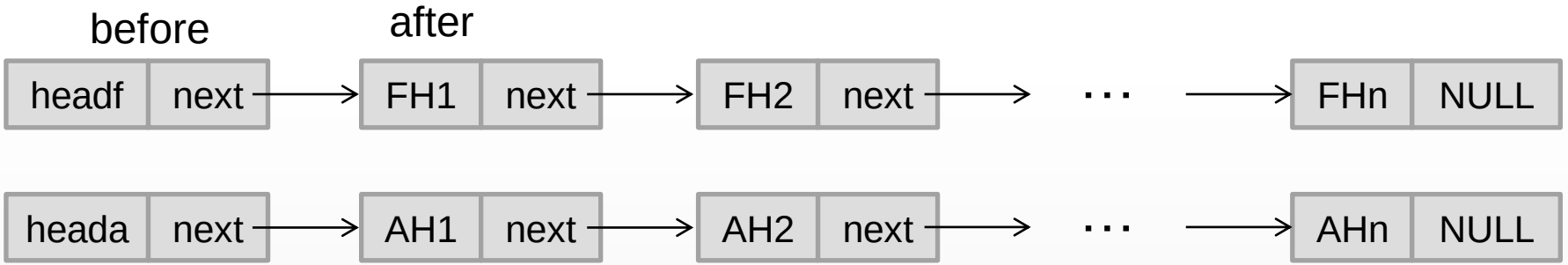
- Task
  - The allocation process alloc( ) and the recycling process free( ) for dynamic partition are respectively implemented based on the first-fit algorithm and the best-fit algorithm in C language.
  - Free holes are managed through a free hole chain.
  - When allocating memory, the system gives priority to using the lower space of the free area.
  - Shows the status of free hole chains after each allocation and recycling.

Experiment 9 Project4

- Free and allocated hole chain
- struct node // node for the chain
- {
- int address, size;
- struct node *next;
- };
- typedef struct node RECT;
- Example

Experiment 9 Project4

- Free and allocated hole chain
- Free hole(FH)
- Allocated hole(AH)



Experiment 9 Project4

- Pseudocode for traversing free hole chain for first-fit
- before=headf;
- after=headf->next;
- while(after!=NULL){
  - if(after->size > the requested size){
    allocate;
    break;
    }else if(after->size == the requested size){
    allocate;
    before-next=after->next;
    free(after);
    }else{
    before=before->next;
    after=after->next;
    }
  }

Experiment 9 Project4

- Pseudocode for traversing free hole chain to reclaim memory
- before=headf;
- after=headf->next;
- back=malloc(sizeof(RECT));
- back->address=the reclaimed address;
- back->size=the reclaimed size;
- flag=false;

Experiment 9 Project4

- while(!flag){
  - If(after==NULL){
  - before->next=back;
  - back-next=after;
  - flag=true;
  - break;
  - }

Experiment 9 Project4

```
if(before->address+before->size==back->address){
    if(back->address+back->size==after->address){// merge back with before and after
      if(before!=headf){
        before->size=before->size+back->size+after->size;
        before->next=after->next;
        free(after);
        free(back);
        flag=true;
        break;
      }
    }else{ //merge back with before
        if(before!=headf){
          before->size=before->size+back->size;
          free(back);
          flag=true;
          break;
        }
     }
}
```

Experiment 9 Project4

```
if(back->address+back->size==after->address){// merge back with after
        after->size=after->size+back->size;
        after->address=back->address;
        free(back);
        flag=true;
        break;
}
if((before->address<=back->address)&&(after->address>back->address)){
    // insert back to the free hole chain
   before->next=back;
   back->next=after;
   flag=true;
}else{
   before=before->next;
   after=after->next;
}
}//while
```

Experiment 9 Project4

- Pseudocode for traversing allocated hole chain to reclaim memory
- before=heada;
- after=heada->next;
- while(after!=NULL){
  - If(after->size == the reclaimed size && after->address==the reclaimed address){
    before->next=after->next;
    free(after);
    break;
    }else{
    before=before->next;
    after=after->next;
    }
  }

Experiment 9 Project4

- Another request sequence:
  - •job1 applies for 130KB
  - •job2 applies for 60KB
  - •job3 applies for 100KB
  - •job1 releases 130KB
  - •job4 applies for 200KB
  - •job3 releases 100KB
  - •job5 applies for 150KB
  - •job1 releases 60KB
  - •job6 applies for 290KB
  - •job5 releases 150KB
  - •job6 releases 290KB
  - •job4 releases 200KB