Formal Development of a Real-Time Operating System Memory Manager

Wen Su and Jean-Raymond Abrial

Contents

0.1 C Code of the Memory Manager

```
#include <stdio.h>

/* CONSTANTS */

/* size of memory (from 0 to 11) */

#define m 12

/* number of free block partitions (from 0 to 4) */

#define d 5
#define TRUE 1
#define FALSE 0
#define NIL -1

/* Linking block sizes to free block partition (0 and 11 are meaningless) */
/* 1..2: partition 0, 3..5: partition 1, 6..7: partition 2, etc. */

/* lower size of each free block partition */

static int lower[d] = {1,3,6,8,10};

/* upper size of each free block partition */

static int upper[d] = {2,5,7,9,10};

/*
 Formal definition of g(q) and g_srh are as below:

 g(q) = min({i| i \in 1..d \and q<=upper(i)})
 g_srh(q) = min({i|i \in 1..d \and q<=lower(i)})
 upper[g[q]] >= q
```

```
 lower[g_srh[q]] >= q
*/

static int g[m] = {0,0,0,1,1,1,2,2,3,3,4,4};
static int g_srh[m] = {0,0,1,1,2,2,2,3,3,4,4,4};

/* VARIABLES */

/* size of blocks (-1:meaningless) */

int size[m] = {1,10,-1,-1,-1,-1,-1,-1,-1,-1,-1,1};

/* pointing to first element (if it exists) of each free block partition chain*/

int f[d] = {-1, -1, -1, -1, 1};

/* left block of a block (-1:meaningless) */

int left[m] = {-1,0,-1,-1,-1,-1,-1,-1,-1,-1,-1,1};

/* next element in doubly linked chain of a partition (-1:meaningless) */

int next[m] = {-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};

/* previous element in doubly linked chain of a partition (-1:meaningless) */

int prev[m] = {-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};

/* free bit of blocks (0:not free, 1:free, -1:meaningless) */

int free_bit[m] = {0,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,0};

/*int search_bit = FALSE;*/
/*int q_loc_0=0;*/

/* PRINTING FUNCTIONS. These functions are added here to visualise
   some results. They are NOT part of the final product */

/* printing size of each block */

int prsize(){
    int n;
    printf("size = ");
    for(n=0; n<=11 ; n++) {
        if (size[n]!=-1){
            printf("%d->%d ",n,size[n]);
        }
```

```
    }
    printf("\n");
    return(0);
}

/* printing left block of each block */

int prleft(){
    int n;
    printf("left = ");
    for(n=0; n<=11 ; n++) {
        if (left[n]!=-1){
            printf("%d->%d ",n,left[n]);
        }
    }
    printf("\n");
    return(0);
}

/* printing doubly linked chain of a group */

int prlink(int e){
    int n = f[e];
    printf("%d..%d: ",lower[e],upper[e]);
    if (n==-1){
        printf("empty");
    } else {
        printf("%d",n);
        while (n!=-1) {
            if (next[n]!=-1)printf("<->%d",next[n]);
            n=next[n];
        }
    }
    printf("\n");
    return(0);
}
/* printing doubly linked chain of all groups */

int prlnnk(){
    int n;
    for (n=0 ; n<=4; n++){
        prlink(n);
    }
    return(0);
}

/* printing free bit of each block */
```

```
int prfree_bit(){
    int n;
    printf("free_bit = ");
    for(n=0; n<=11 ; n++) {
        if (free_bit[n]!=-1) {
            printf("%d->%d ",n,free_bit[n]);
        }
    }
    printf("\n");
    return(0);
}

/* printing all data */

int printdata(){
    prsize();
    prfree_bit();
    prleft();
    prlnnk();
    printf("\n");
    return(0);
}

/* ENCAPSULATING FUNCTIONS (as in the EventB-Rodin development).
   These functions are hand translation of the last refinement
   of the Rodin development */

int remove_from_free(int b){
    free_bit[b] = FALSE;
    if (f[g[size[b]]]!=b)
        next[prev[b]]=next[b];
    else
        f[g[size[b]]]=next[b];
    if (next[b]!=NIL) prev[next[b]]=prev[b];
    next[b] = -1;
    prev[b] = -1;
    return(0);
}

int make_free(int b){
    int a=f[g[size[b]]];
    free_bit[b] = TRUE;
    f[g[size[b]]] = b;
    next[b]=a;
    prev[b]=NIL;
    if (a!=NIL) prev[a] = b;
```

```
    return(0);
}

int reduce_create(int b,int q){
    size[b+q]=size[b]-q;
    left[b+q]=b;
    free_bit[b+q]=FALSE;
    left[b+size[b]]=b+q;
    size[b]=q;
    return(0);
}

int merge_right(int b){
    int s;
    s=size[b];
    left[b+s+size[b+s]]=b;
    size[b]=s+size[b+s];
    size[b+s]=-1;
    left[b+s]=-1;
    free_bit[b+s]=-1;
    return(0);
}

/* EXTERNAL FUNCTIONS (as in the EventB-Rodin development).
   These functions are hand translation of the last refinement
   of the Rodin development. We added a few printing instructions,
   which should be REMOVED in the final product */

int search(int q0){
    int bloc_0=-1;
    if (q0>=1 && q0<=m-2) {
        for (int i=g_srh[q0] ; i<=d-1; i++){
            if (f[i]!=-1){
                bloc_0=f[i];
                break;
            }
        }
        if (bloc_0==-1) {
            printf("FAILURE \n\n");
        }
    }else{
        printf("FAILURE \n\n");
    }
    return bloc_0;
}

int allocate_mem(int q){
```

```
    printf("allocate_mem(%d)\n",q);
    int bloc_0=search(q);
    if (bloc_0!=-1) {
        if (q<size[bloc_0]){
            remove_from_free(bloc_0);
            reduce_create(bloc_0, q);
            make_free(bloc_0+q);
        }else if (q==size[bloc_0]){
            remove_from_free(bloc_0);
        }
        printf("SUCCESS ");
        printf("block=%d\n\n",bloc_0);
    }
    printdata();
    return(0);
}

int free_mem(int b){
    int bloc;
    printf("free_mem(%d)\n",b);
    if (size[b]!=-1 && free_bit[b]==0 && b!=0 && b!=m+1 && size[left[b]]!=-1) {
        if (free_bit[left[b]]==TRUE) {
            bloc=b;
            remove_from_free(left[b]);
            bloc = left[bloc];
            merge_right(bloc);
        }else{
            bloc=b;
        }
        if (free_bit[bloc+size[bloc]]==TRUE) {
            remove_from_free(bloc+size[bloc]);
            merge_right(bloc);
        }
        make_free(bloc);
        printf("SUCCESS\n\n");
    }else{
        printf("FAILURE\n\n");
    }
    printdata();
    return(0);
}

/* MAIN FUNCTION. The instructions of this function correspond to some
   test executions. This function is NOT part of the final product */

int main(int argc, const char * argv[]){
    printdata();
```

```
    allocate_mem(6);
    allocate_mem(15);
    allocate_mem(10);
    allocate_mem(2);
    allocate_mem(3);
    free_mem(1);
    allocate_mem(1);
    free_mem(7);
    allocate_mem(3);
    allocate_mem(5);
    free_mem(9);
    free_mem(1);
    free_mem(4);
    free_mem(4);
    allocate_mem(15);
    allocate_mem(1);
    allocate_mem(1);
    allocate_mem(1);
    allocate_mem(1);
    allocate_mem(1);
    allocate_mem(1);
    allocate_mem(1);
    allocate_mem(1);
    free_mem(1);
    free_mem(3);
    free_mem(5);
    free_mem(7);
    free_mem(2);
    free_mem(4);
    free_mem(6);
    free_mem(8);
}
```

0.2 Output of Execution

```
size = 0->1 1->10 11->1
free_bit = 0->0 1->1 11->0
left = 1->0 11->1
1..2: empty
3..5: empty
6..7: empty
8..9: empty
10..10: 1

allocate_mem(6)
SUCCESS block=1
```

```
size = 0->1 1->6 7->4 11->1
free_bit = 0->0 1->0 7->1 11->0
left = 1->0 7->1 11->7
1..2: empty
3..5: 7
6..7: empty
8..9: empty
10..10: empty

allocate_mem(15)
FAILURE

size = 0->1 1->6 7->4 11->1
free_bit = 0->0 1->0 7->1 11->0
left = 1->0 7->1 11->7
1..2: empty
3..5: 7
6..7: empty
8..9: empty
10..10: empty

allocate_mem(10)
FAILURE

size = 0->1 1->6 7->4 11->1
free_bit = 0->0 1->0 7->1 11->0
left = 1->0 7->1 11->7
1..2: empty
3..5: 7
6..7: empty
8..9: empty
10..10: empty

allocate_mem(2)
SUCCESS block=7

size = 0->1 1->6 7->2 9->2 11->1
free_bit = 0->0 1->0 7->0 9->1 11->0
left = 1->0 7->1 9->7 11->9
1..2: 9
3..5: empty
6..7: empty
8..9: empty
10..10: empty

allocate_mem(3)
FAILURE
```

```
size = 0->1 1->6 7->2 9->2 11->1
free_bit = 0->0 1->0 7->0 9->1 11->0
left = 1->0 7->1 9->7 11->9
1..2: 9
3..5: empty
6..7: empty
8..9: empty
10..10: empty

free_mem(1)
SUCCESS

size = 0->1 1->6 7->2 9->2 11->1
free_bit = 0->0 1->1 7->0 9->1 11->0
left = 1->0 7->1 9->7 11->9
1..2: 9
3..5: empty
6..7: 1
8..9: empty
10..10: empty

allocate_mem(1)
SUCCESS block=9

size = 0->1 1->6 7->2 9->1 10->1 11->1
free_bit = 0->0 1->1 7->0 9->0 10->1 11->0
left = 1->0 7->1 9->7 10->9 11->10
1..2: 10
3..5: empty
6..7: 1
8..9: empty
10..10: empty

free_mem(7)
SUCCESS

size = 0->1 1->8 9->1 10->1 11->1
free_bit = 0->0 1->1 9->0 10->1 11->0
left = 1->0 9->1 10->9 11->10
1..2: 10
3..5: empty
6..7: empty
8..9: 1
10..10: empty

allocate_mem(3)
```

SUCCESS block=1

```
size = 0->1 1->3 4->5 9->1 10->1 11->1
free_bit = 0->0 1->0 4->1 9->0 10->1 11->0
left = 1->0 4->1 9->4 10->9 11->10
1..2: 10
3..5: 4
6..7: empty
8..9: empty
10..10: empty
```

```
allocate_mem(5)
```
FAILURE

```
size = 0->1 1->3 4->5 9->1 10->1 11->1
free_bit = 0->0 1->0 4->1 9->0 10->1 11->0
left = 1->0 4->1 9->4 10->9 11->10
1..2: 10
3..5: 4
6..7: empty
8..9: empty
10..10: empty
```

```
free_mem(9)
```
SUCCESS

```
size = 0->1 1->3 4->7 11->1
free_bit = 0->0 1->0 4->1 11->0
left = 1->0 4->1 11->4
1..2: empty
3..5: empty
6..7: 4
8..9: empty
10..10: empty
```

```
free_mem(1)
```
SUCCESS

```
size = 0->1 1->10 11->1
free_bit = 0->0 1->1 11->0
left = 1->0 11->1
1..2: empty
3..5: empty
6..7: empty
8..9: empty
10..10: 1
```

```
free_mem(4)
FAILURE

size = 0->1 1->10 11->1
free_bit = 0->0 1->1 11->0
left = 1->0 11->1
1..2: empty
3..5: empty
6..7: empty
8..9: empty
10..10: 1

free_mem(4)
FAILURE

size = 0->1 1->10 11->1
free_bit = 0->0 1->1 11->0
left = 1->0 11->1
1..2: empty
3..5: empty
6..7: empty
8..9: empty
10..10: 1

allocate_mem(15)
FAILURE

size = 0->1 1->10 11->1
free_bit = 0->0 1->1 11->0
left = 1->0 11->1
1..2: empty
3..5: empty
6..7: empty
8..9: empty
10..10: 1

allocate_mem(1)
SUCCESS block=1

size = 0->1 1->1 2->9 11->1
free_bit = 0->0 1->0 2->1 11->0
left = 1->0 2->1 11->2
1..2: empty
3..5: empty
6..7: empty
8..9: 2
10..10: empty
```

```
allocate_mem(1)
SUCCESS block=2

size = 0->1 1->1 2->1 3->8 11->1
free_bit = 0->0 1->0 2->0 3->1 11->0
left = 1->0 2->1 3->2 11->3
1..2: empty
3..5: empty
6..7: empty
8..9: 3
10..10: empty

allocate_mem(1)
SUCCESS block=3

size = 0->1 1->1 2->1 3->1 4->7 11->1
free_bit = 0->0 1->0 2->0 3->0 4->1 11->0
left = 1->0 2->1 3->2 4->3 11->4
1..2: empty
3..5: empty
6..7: 4
8..9: empty
10..10: empty

allocate_mem(1)
SUCCESS block=4

size = 0->1 1->1 2->1 3->1 4->1 5->6 11->1
free_bit = 0->0 1->0 2->0 3->0 4->0 5->1 11->0
left = 1->0 2->1 3->2 4->3 5->4 11->5
1..2: empty
3..5: empty
6..7: 5
8..9: empty
10..10: empty

allocate_mem(1)
SUCCESS block=5

size = 0->1 1->1 2->1 3->1 4->1 5->1 6->5 11->1
free_bit = 0->0 1->0 2->0 3->0 4->0 5->0 6->1 11->0
left = 1->0 2->1 3->2 4->3 5->4 6->5 11->6
1..2: empty
3..5: 6
6..7: empty
8..9: empty
```

```
10..10: empty

allocate_mem(1)
SUCCESS block=6

size = 0->1 1->1 2->1 3->1 4->1 5->1 6->1 7->4 11->1
free_bit = 0->0 1->0 2->0 3->0 4->0 5->0 6->0 7->1 11->0
left = 1->0 2->1 3->2 4->3 5->4 6->5 7->6 11->7
1..2: empty
3..5: 7
6..7: empty
8..9: empty
10..10: empty

allocate_mem(1)
SUCCESS block=7

size = 0->1 1->1 2->1 3->1 4->1 5->1 6->1 7->1 8->3 11->1
free_bit = 0->0 1->0 2->0 3->0 4->0 5->0 6->0 7->0 8->1 11->0
left = 1->0 2->1 3->2 4->3 5->4 6->5 7->6 8->7 11->8
1..2: empty
3..5: 8
6..7: empty
8..9: empty
10..10: empty

allocate_mem(1)
SUCCESS block=8

size = 0->1 1->1 2->1 3->1 4->1 5->1 6->1 7->1 8->1 9->2 11->1
free_bit = 0->0 1->0 2->0 3->0 4->0 5->0 6->0 7->0 8->0 9->1 11->0
left = 1->0 2->1 3->2 4->3 5->4 6->5 7->6 8->7 9->8 11->9
1..2: 9
3..5: empty
6..7: empty
8..9: empty
10..10: empty

free_mem(1)
SUCCESS

size = 0->1 1->1 2->1 3->1 4->1 5->1 6->1 7->1 8->1 9->2 11->1
free_bit = 0->0 1->1 2->0 3->0 4->0 5->0 6->0 7->0 8->0 9->1 11->0
left = 1->0 2->1 3->2 4->3 5->4 6->5 7->6 8->7 9->8 11->9
1..2: 1<->9
3..5: empty
6..7: empty
```

```
8..9: empty
10..10: empty

free_mem(3)
SUCCESS

size = 0->1 1->1 2->1 3->1 4->1 5->1 6->1 7->1 8->1 9->2 11->1
free_bit = 0->0 1->1 2->0 3->1 4->0 5->0 6->0 7->0 8->0 9->1 11->0
left = 1->0 2->1 3->2 4->3 5->4 6->5 7->6 8->7 9->8 11->9
1..2: 3<->1<->9
3..5: empty
6..7: empty
8..9: empty
10..10: empty

free_mem(5)
SUCCESS

size = 0->1 1->1 2->1 3->1 4->1 5->1 6->1 7->1 8->1 9->2 11->1
free_bit = 0->0 1->1 2->0 3->1 4->0 5->1 6->0 7->0 8->0 9->1 11->0
left = 1->0 2->1 3->2 4->3 5->4 6->5 7->6 8->7 9->8 11->9
1..2: 5<->3<->1<->9
3..5: empty
6..7: empty
8..9: empty
10..10: empty

free_mem(7)
SUCCESS

size = 0->1 1->1 2->1 3->1 4->1 5->1 6->1 7->1 8->1 9->2 11->1
free_bit = 0->0 1->1 2->0 3->1 4->0 5->1 6->0 7->1 8->0 9->1 11->0
left = 1->0 2->1 3->2 4->3 5->4 6->5 7->6 8->7 9->8 11->9
1..2: 7<->5<->3<->1<->9
3..5: empty
6..7: empty
8..9: empty
10..10: empty

free_mem(2)
SUCCESS

size = 0->1 1->3 4->1 5->1 6->1 7->1 8->1 9->2 11->1
free_bit = 0->0 1->1 4->0 5->1 6->0 7->1 8->0 9->1 11->0
left = 1->0 4->1 5->4 6->5 7->6 8->7 9->8 11->9
1..2: 7<->5<->9
3..5: 1
```

```
6..7: empty
8..9: empty
10..10: empty

free_mem(4)
SUCCESS

size = 0->1 1->5 6->1 7->1 8->1 9->2 11->1
free_bit = 0->0 1->1 6->0 7->1 8->0 9->1 11->0
left = 1->0 6->1 7->6 8->7 9->8 11->9
1..2: 7<->9
3..5: 1
6..7: empty
8..9: empty
10..10: empty

free_mem(6)
SUCCESS

size = 0->1 1->7 8->1 9->2 11->1
free_bit = 0->0 1->1 8->0 9->1 11->0
left = 1->0 8->1 9->8 11->9
1..2: 9
3..5: empty
6..7: 1
8..9: empty
10..10: empty

free_mem(8)
SUCCESS

size = 0->1 1->10 11->1
free_bit = 0->0 1->1 11->0
left = 1->0 11->1
1..2: empty
3..5: empty
6..7: empty
8..9: empty
10..10: 1
```