Name: Krithika Swaminathan
Roll No.: 205001057

Assignment 8 – Implementation of Memory Allocation Techniques: First Fit, Best Fit and Worst Fit

Date: 23/05/2022

Aim:

To implement the following memory allocation techniques:

1. First fit
2. Best fit
3. Worst fit

Algorithm:

1. Start
2. Get the required details on the partitioning of the physical memory such as:
    1. The number of partitions
    2. Starting addresses
    3. Ending addresses
3. Calculate the size of each partition and the free space available in each partition.
4. Display a menu with options for the three memory allocation techniques:
    5. First fit
    6. Best fit
    7. Worst fit
    8. For each technique, display a menu with options for allocation, deallocation, displaying the memory and for coalescing of holes.
9. Stop

1. *First fit:*
    1. Start
    2. To allocate a process:
        1. Read the process ID and the size of the process as input from the user.
        2. Iterate through the partitions until a partition with enough free space to accommodate the process is found.
        3. Insert the process into the physical memory and update the allocated and free memory accordingly.
        4. If no free space is available, print an error message indicating the same.
    3. To deallocate a process:
        1. Read the process ID of the process to be deallocated as input from the user.
        2. If the process is found in the physical memory, remove the process and create a hole in the memory space that the process was occupying.

3. If the process is not found in the physical memory, print an error message indicating the same.

4. To merge holes:
    1. If a partition contains a hole and the partitions immediately following it also contain holes, then note the starting address of the partition and the ending address of the last partition to contain a hole in this sequence.
    2. Set the newly acquired starting address and ending address as the starting and ending of a single partition that replaces the entire sequence of partitions containing holes.
    3. Repeat this strategy for all the sequences of holes in the physical memory.
5. Stop

2. Best fit:

1. Start
2. To allocate a process:
    1. Read the process ID and the size of the process as input from the user.
    2. Iterate through the partitions until a partition with the minimum free space to accommodate the process is found.
    3. Insert the process into the physical memory and update the allocated and free memory accordingly.
    4. If no free space is available, print an error message indicating the same.
3. To deallocate a process:
    1. Read the process ID of the process to be deallocated as input from the user.
    2. If the process is found in the physical memory, remove the process and create a hole in the memory space that the process was occupying.
    3. If the process is not found in the physical memory, print an error message indicating the same.
4. To merge holes:
    1. If a partition contains a hole and the partitions immediately following it also contain holes, then note the starting address of the partition and the ending address of the last partition to contain a hole in this sequence.
    2. Set the newly acquired starting address and ending address as the starting and ending of a single partition that replaces the entire sequence of partitions containing holes.
    3. Repeat this strategy for all the sequences of holes in the physical memory.
5. Stop

3. *Worst fit:*
    1. Start
    2. To allocate a process:
        1. Read the process ID and the size of the process as input from the user.
        2. Iterate through the partitions until a partition with the maximum free space to accommodate the process is found.
        3. Insert the process into the physical memory and update the allocated and free memory accordingly.
        4. If no free space is available, print an error message indicating the same.
    3. To deallocate a process:
        1. Read the process ID of the process to be deallocated as input from the user.
        2. If the process is found in the physical memory, remove the process and create a hole in the memory space that the process was occupying.
        3. If the process is not found in the physical memory, print an error message indicating the same.
    4. To merge holes:
        1. If a partition contains a hole and the partitions immediately following it also contain holes, then note the starting address of the partition and the ending address of the last partition to contain a hole in this sequence.
        2. Set the newly acquired starting address and ending address as the starting and ending of a single partition that replaces the entire sequence of partitions containing holes.
        3. Repeat this strategy for all the sequences of holes in the physical memory.
    5. Stop

Programs:

Code:

```
// Program to implement Memory Management Algorithms - First fit, Best fit and Worst fit

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX 10

typedef struct Memnode {
  int start, end, size, freesize;
        char status[3];
        } mem;

void reset (mem space1[MAX], mem space2[MAX], int n) {
```

```
	for (int i=0; i<n; i++) {
		space1[i].start = space2[i].start;
		space1[i].end = space2[i].end;
		space1[i].size = space2[i].size;
		space1[i].freesize = space2[i].freesize;
		strcpy(space1[i].status,space2[i].status);
		}
	}

void shiftl (mem space[MAX], int pos, int n) {
	for (int i=pos; i<n; i++) {
		space[i].start = space[i+1].start;
		space[i].end = space[i+1].end;
		space[i].size = space[i+1].size;
		space[i].freesize = space[i+1].freesize;
		strcpy(space[i].status,space[i+1].status);
		}
	}

void displayfree (mem space[MAX], int numP, int allocs) {
	int n = numP-allocs;
	printf("\n");
	if (n==0) {
		printf("NULL\n");
		return;
		}
	for (int i=0; i<n; i++)
		printf("----------------");
	printf("\n");
	for (int i=0; i<numP; i++) {
		if (strcmp(space[i].status,"H")==0)
			printf("|\t\t%s\t\t|",space[i].status);
		}
	printf("\n");
	for (int i=0; i<n; i++)
		printf("----------------");
	printf("\n");
	for (int i=0; i<numP; i++) {
		if (strcmp(space[i].status,"H")==0)
			printf(" %d\t\t %d",space[i].start,space[i].end);
		}
	printf("\n");
	}

void display (mem space[MAX], int n) {
	printf("\n");
	if (n==0) {
		printf("NULL\n");
		return;
```

```
                }
        for (int i=0; i<n; i++)
                printf("----------------");
        printf("\n");
        for (int i=0; i<n; i++) {
                printf("|\t\t%s\t\t|",space[i].status);
                }
        printf("\n");
        for (int i=0; i<n; i++)
                printf("----------------");
        printf("\n");
        for (int i=0; i<n; i++) {
                printf(" %d\t\t %d",space[i].start,space[i].end);
                }
        printf("\n");
        }

int main() {
        printf("\n__MEMORY MANAGEMENT ALGORITHMS__\n");

        mem ph[MAX], free[MAX], alloc[MAX], temp[MAX];

        int numP;
        printf("\nEnter the no. of partitions in memory: ");
        scanf("%d",&numP);

        for (int i=0; i<numP; i++) {
                printf("\nPartition %d: ",i+1);
                printf("\n\tEnter starting address: ");
                scanf("%d",&ph[i].start);
                printf("\tEnter ending address: ");
                scanf("%d",&ph[i].end);
                ph[i].size = ph[i].end - ph[i].start;
                ph[i].freesize = ph[i].size;
                strcpy(ph[i].status,"H");
                }

        reset(temp,ph,numP);
        //displaying physical memory
        printf("\nPhysical memory: \n");
        display(ph,numP);
        //displaying copy of physical memory
        /*printf("\nTemp memory: \n");
        display(temp,numP);*/

        int algoch;
        do {
                printf("\nMemory allocation: \n1. First Fit\n2. Best Fit\n3. Worst Fit\n4. Exit from
program\n");
```

```
		printf("Enter choice: ");
		scanf("%d",&algoch);

		switch(algoch) {
			case 1: {
				printf("\n__First Fit Memory Allocation Algorithm__\n");
				reset(ph,temp,numP);
				reset(free,ph,numP);
				int acount = 0, pcount = 0, numPh = numP, de = 0;  //counting
allocation partitions in memory

				int choice;
				do {
					printf("\nMENU: \n1. Allocate\n2. Deallocate\n3. Display\n4.
Coalescing of holes\n5. Back to program\n");
					printf("Enter choice: ");
					scanf("%d",&choice);

					switch(choice) {
						case 1: {
							printf("\nAllocating memory: \n");
							char pid[3]; int psize;
							printf("Enter process ID: ");
							scanf("%s",pid);
							printf("Enter process size: ");
							scanf("%d",&psize);

							for (int i=0; i<numPh; i++) {
								if (psize <= free[i].freesize) {
									alloc[acount].start = free[i].start;
									alloc[acount].end =
alloc[acount].start + psize;

									alloc[acount].size = psize;
									alloc[acount].freesize = 0;
									strcpy(alloc[acount].status,pid);
									free[i].start += psize;
									free[i].size -= psize;
									free[i].freesize -= psize;
									ph[pcount].freesize -= psize;
									numPh++;
									pcount = i+acount+de;
									for (int j=numPh-1; j>pcount; j--)
{

										ph[j].start = ph[j-1].start;
										ph[j].end = ph[j-1].end;
										ph[j].freesize = ph[j-
1].freesize;
										strcpy(ph[j].status,ph[j-
1].status);
```

```
                                        }
                                ph[pcount+1].start = free[i].start;
                                ph[pcount].end =
alloc[acount].end;
                                ph[pcount].start =
alloc[acount].start;
                                ph[pcount].freesize = 0;

        strcpy(ph[pcount].status,alloc[acount].status);

                                acount++;
                                break;
                                }
                        }

                break;
                }

        case 2: {
                printf("\nDeallocating memory: \n");
                char pid[3]; int psize;
                printf("Enter process ID: ");
                scanf("%s",pid);

                int ploc = -1, floc, aloc = -1;
                for (int i=0; i<numPh; i++) {
                        if (strcmp(ph[i].status,pid)==0)
                                ploc = i;
                        }
                for (int i=0; i<numPh; i++) {
                        if (strcmp(alloc[i].status,pid)==0)
                                aloc = i;
                        }
                floc = ploc-acount+1;
                psize = alloc[aloc].size;

                free[floc].start -= psize;
                free[floc].freesize += psize;

                shiftl(alloc,aloc,acount);
                acount--;

                strcpy(ph[ploc].status,"H");
                ph[ploc].freesize = psize;

                de++;

                break;
                }
```

```
                        case 3: {
                                printf("\nDisplaying memory: \n");

                                printf("\nAllocated memory: \n");
                                display(alloc,acount);
                                printf("\nFree memory: \n");
                                displayfree(ph,numPh,acount);
                                printf("\nPhysical memory: \n");
                                display(ph,numPh);

                                break;
                                }

                        case 4: {
                                printf("\nMerged holes! \n");
                                int start = 0, end = 0;
                                for (int i=1; i<numPh; i++) {
                                        if (strcmp(ph[i-1].status,"H")==0) {
                                                start = ph[i-1].start;
                                                int j = i;
                                                while
(strcmp(ph[j].status,"H")==0 && j<numPh) {
                                                        shiftl(ph,j-1,numPh-1);
                                                        numPh--;
                                                        }
                                                end = ph[i].end;
                                                ph[i-1].start = start;
                                                ph[i-1].end = end;
                                                }
                                        }

                                break;
                                }

                        case 5: {
                                printf("\nReturning to main program...\n");
                                break;
                                }

                        default: {
                                printf("\nInvalid choice! Try again.\n");
                                }
                        }
                } while(choice!=5);

        break;
        }
```

```
                case 2: {
                        printf("\n__Best Fit Memory Allocation Algorithm__\n");
                        reset(ph,temp,numP);
                        reset(free,ph,numP);
                        int acount = 0, pcount = 0, numPh = numP, de = 0;  //counting
allocation partitions in memory

                        int choice;
                        do {
                                printf("\nMENU: \n1. Allocate\n2. Deallocate\n3. Display\n4.
Coalescing of holes\n5. Back to program\n");
                                printf("Enter choice: ");
                                scanf("%d",&choice);

                                switch(choice) {
                                        case 1: {
                                                printf("\nAllocating memory: \n");
                                                char pid[3]; int psize;
                                                printf("Enter process ID: ");
                                                scanf("%s",pid);
                                                printf("Enter process size: ");
                                                scanf("%d",&psize);

                                                int min_idx = 0;
                                                for (int i=0; i<numP; i++) {
                                                        if (free[i].freesize <
free[min_idx].freesize && psize <= free[i].freesize)
                                                                min_idx = i;
                                                        }

                                                if (psize <= free[min_idx].freesize) {
                                                                alloc[acount].start =
free[min_idx].start;
                                                                alloc[acount].end =
alloc[acount].start + psize;
                                                                alloc[acount].size = psize;
                                                                alloc[acount].freesize = 0;
                                                                strcpy(alloc[acount].status,pid);
                                                                free[min_idx].start += psize;
                                                                free[min_idx].size -= psize;
                                                                free[min_idx].freesize -= psize;
                                                                numPh++;
                                                                pcount = min_idx+acount+de;
                                                                for (int j=numPh-1; j>pcount; j--)
{
                                                                        ph[j].start = ph[j-1].start;
                                                                        ph[j].end = ph[j-1].end;
                                                                        ph[j].freesize = ph[j-
1].freesize;
```

```
                                        strcpy(ph[j].status,ph[j-
1].status);
                                        }
                                ph[pcount+1].start =
free[min_idx].start;
                                ph[pcount].end =
alloc[acount].end;
                                ph[pcount].start =
alloc[acount].start;
                                ph[pcount].freesize = 0;

        strcpy(ph[pcount].status,alloc[acount].status);

                                acount++;
                                break;
                                }

                        break;
                        }

                case 2: {
                        printf("\nDeallocating memory: \n");
                        char pid[3]; int psize;
                        printf("Enter process ID: ");
                        scanf("%s",pid);

                        int ploc = -1, floc, aloc = -1;
                        for (int i=0; i<numPh; i++) {
                                if (strcmp(ph[i].status,pid)==0)
                                        ploc = i;
                                }
                        for (int i=0; i<numPh; i++) {
                                if (strcmp(alloc[i].status,pid)==0)
                                        aloc = i;
                                }
                        floc = ploc-acount+1;
                        psize = alloc[aloc].size;

                        free[floc].start -= psize;
                        free[floc].freesize += psize;

                        shiftl(alloc,aloc,acount);
                        acount--;

                        strcpy(ph[ploc].status,"H");
                        ph[ploc].freesize = psize;

                        de++;
```

```
                        break;
                        }

                case 3: {
                        printf("\nDisplaying memory: \n");

                        printf("\nAllocated memory: \n");
                        display(alloc,acount);
                        printf("\nFree memory: \n");
                        displayfree(ph,numPh,acount);
                        printf("\nPhysical memory: \n");
                        display(ph,numPh);

                        break;
                        }

                case 4: {
                        printf("\nMerged holes! \n");
                        int start = 0, end = 0;
                        for (int i=1; i<numPh; i++) {
                                if (strcmp(ph[i-1].status,"H")==0) {
                                        start = ph[i-1].start;
                                        int j = i;
                                        while
(strcmp(ph[j].status,"H")==0 && j<numPh) {
                                                shiftl(ph,j-1,numPh-1);
                                                numPh--;
                                                }
                                        end = ph[i].end;
                                        ph[i-1].start = start;
                                        ph[i-1].end = end;
                                        }
                                }

                        break;
                        }

                case 5: {
                        printf("\nReturning to main program...\n");
                        break;
                        }

                default: {
                        printf("\nInvalid choice! Try again.\n");
                        }
                }
        } while(choice!=5);

break;
```

```
                }

            case 3: {
                printf("\n__Worst Fit Memory Allocation Algorithm__\n");
                reset(ph,temp,numP);
                reset(free,ph,numP);
                int acount = 0, pcount = 0, numPh = numP, de = 0;  //counting
allocation partitions in memory

                int choice;
                do {
                    printf("\nMENU: \n1. Allocate\n2. Deallocate\n3. Display\n4.
Coalescing of holes\n5. Back to program\n");
                    printf("Enter choice: ");
                    scanf("%d",&choice);

                    switch(choice) {
                        case 1: {
                            printf("\nAllocating memory: \n");
                            char pid[3]; int psize;
                            printf("Enter process ID: ");
                            scanf("%s",pid);
                            printf("Enter process size: ");
                            scanf("%d",&psize);

                            int max_idx = 0;
                            for (int i=0; i<numP; i++) {
                                if (free[i].freesize >
free[max_idx].freesize && psize <= free[i].freesize)
                                    max_idx = i;
                                }

                            if (psize <= free[max_idx].freesize) {
                                alloc[acount].start = free[max_idx].start;
                                alloc[acount].end = alloc[acount].start +
psize;

                                alloc[acount].size = psize;
                                alloc[acount].freesize = 0;
                                strcpy(alloc[acount].status,pid);
                                free[max_idx].start += psize;
                                free[max_idx].size -= psize;
                                free[max_idx].freesize -= psize;
                                ph[max_idx].freesize -= psize;
                                numPh++;
                                pcount = max_idx+acount+de;
                                for (int j=numPh-1; j>pcount; j--) {
                                    ph[j].start = ph[j-1].start;
                                    ph[j].end = ph[j-1].end;
                                    ph[j].freesize = ph[j-1].freesize;
```

```
                                        strcpy(ph[j].status,ph[j-1].status);
                                        }
                                ph[pcount+1].start = free[max_idx].start;
                                ph[pcount].end = alloc[acount].end;
                                ph[pcount].start = alloc[acount].start;
                                ph[pcount].freesize = 0;

strcpy(ph[pcount].status,alloc[acount].status);

                                acount++;
                                break;
                                }

                        break;
                        }

                case 2: {
                        printf("\nDeallocating memory: \n");
                        char pid[3]; int psize;
                        printf("Enter process ID: ");
                        scanf("%s",pid);

                        int ploc = -1, floc, aloc = -1;
                        for (int i=0; i<numPh; i++) {
                                if (strcmp(ph[i].status,pid)==0)
                                        ploc = i;
                                }
                        for (int i=0; i<numPh; i++) {
                                if (strcmp(alloc[i].status,pid)==0)
                                        aloc = i;
                                }
                        floc = ploc-acount+1;
                        psize = alloc[aloc].size;

                        free[floc].start -= psize;
                        free[floc].freesize += psize;

                        shiftl(alloc,aloc,acount);
                        acount--;

                        strcpy(ph[ploc].status,"H");
                        ph[ploc].freesize = psize;

                        de++;

                        break;
                        }

                case 3: {
```

```
                                        printf("\nDisplaying memory: \n");

                                        printf("\nAllocated memory: \n");
                                        display(alloc,acount);
                                        printf("\nFree memory: \n");
                                        displayfree(ph,numPh,acount);
                                        printf("\nPhysical memory: \n");
                                        display(ph,numPh);

                                        break;
                                        }

                                case 4: {
                                        printf("\nMerged holes! \n");
                                        int start = 0, end = 0;
                                        for (int i=1; i<numPh; i++) {
                                                if (strcmp(ph[i-1].status,"H")==0) {
                                                        start = ph[i-1].start;
                                                        int j = i;
                                                        while
(strcmp(ph[j].status,"H")==0 && j<numPh) {
                                                                shiftl(ph,j-1,numPh-1);
                                                                numPh--;
                                                                }
                                                        end = ph[i].end;
                                                        ph[i-1].start = start;
                                                        ph[i-1].end = end;
                                                        }
                                                }

                                        break;
                                        }

                                case 5: {
                                        printf("\nReturning to main program...\n");
                                        break;
                                        }

                                default: {
                                        printf("\nInvalid choice! Try again.\n");
                                        }
                                }
                        } while(choice!=5);

                break;
                }

        case 4: {
                printf("\nExiting the program...\n");
```

```
                        exit(0);
                        break;
                        }

                default: {
                        printf("\nInvalid choice! Try again.\n");
                        }
                }

        } while (algoch != 4);

return 0;
}
```

Name: Krithika Swaminathan
Roll No.: 205001057

Output:

```
~/OSL$ ./a8

__MEMORY MANAGEMENT ALGORITHMS__

Enter the no. of partitions in memory: 4

Partition 1:
    Enter starting address: 100
    Enter ending address: 110

Partition 2:
    Enter starting address: 110
    Enter ending address: 117

Partition 3:
    Enter starting address: 117
    Enter ending address: 120

Partition 4:
    Enter starting address: 120
    Enter ending address: 124

Physical memory:

------------------------------------------------------------
|      H      ||      H      ||      H      ||      H      |
------------------------------------------------------------
 100         110 110         117 117         120 120         124
```

```
Memory allocation:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit from program
Enter choice: 1

__First Fit Memory Allocation Algorithm__

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 1

Allocating memory:
Enter process ID: P1
Enter process size: 5

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3

Displaying memory:
```

Name: Krithika Swaminathan
Roll No.: 205001057

```
Allocated memory:

---------------
|      P1     |
---------------
 100        105

Free memory:

-------------------------------------------------------------
|      H      ||      H      ||      H      ||      H      |
-------------------------------------------------------------
 105        110 110        117 117        120 120        124

Physical memory:

-------------------------------------------------------------------------------
|      P1     ||      H      ||      H      ||      H      ||      H      |
-------------------------------------------------------------------------------
 100        105 105        110 110        117 117        120 120        124

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 1
```

```
Allocating memory:
Enter process ID: P2
Enter process size: 6

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3

Displaying memory:

Allocated memory:

--------------------------------
|      P1     ||      P2     |
--------------------------------
 100        105 110        116

Free memory:

-------------------------------------------------------------
|      H      ||      H      ||      H      ||      H      |
-------------------------------------------------------------
 105        110 116        117 117        120 120        124
```

Name: Krithika Swaminathan
Roll No.: 205001057

```
Physical memory:

-------------------------------------------------------------------------------------------
|      P1     ||     H      ||     P2     ||     H      ||     H      ||     H      |
-------------------------------------------------------------------------------------------
 100        105 105        110 110        116 116        117 117        120 120        124

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 2

Deallocating memory:
Enter process ID: P2

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3

Displaying memory:

Allocated memory:
```

```
Displaying memory:

Allocated memory:

---------------
|      P1     |
---------------
 100        105

Free memory:

-------------------------------------------------------------------------
|      H      ||     H      ||     H      ||     H      ||     H      |
-------------------------------------------------------------------------
 105        110 110        116 116        117 117        120 120        124

Physical memory:

-------------------------------------------------------------------------------------------
|      P1     ||     H      ||     H      ||     H      ||     H      ||     H      |
-------------------------------------------------------------------------------------------
 100        105 105        110 110        116 116        117 117        120 120        124

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 4
```

Name: Krithika Swaminathan
Roll No.: 205001057

```
Merged holes!

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3

Displaying memory:

Allocated memory:

---------------
|       P1      |
---------------
 100         105

Free memory:

---------------
|       H       |
---------------
 105         124
```

```
Physical memory:

 --------------------------------
 |       P1      ||      H       |
 --------------------------------
  100         105 105         124

 MENU:
 1. Allocate
 2. Deallocate
 3. Display
 4. Coalescing of holes
 5. Back to program
 Enter choice: 5

 Returning to main program...

 Memory allocation:
 1. First Fit
 2. Best Fit
 3. Worst Fit
 4. Exit from program
 Enter choice: 2

 __Best Fit Memory Allocation Algorithm__

 MENU:
 1. Allocate
 2. Deallocate
 3. Display
```

Name: Krithika Swaminathan
Roll No.: 205001057

```
4. Coalescing of holes
5. Back to program
Enter choice: 1

Allocating memory:
Enter process ID: P1
Enter process size: 6

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3

Displaying memory:

Allocated memory:

---------------
|      P1     |
---------------
 110        116

Free memory:

-------------------------------------------------------------
|      H      ||      H      ||      H      ||      H      |
-------------------------------------------------------------
```

```
Physical memory:

-------------------------------------------------------------------------------
|      H      ||      P1     ||      H      ||      H      ||      H      |
-------------------------------------------------------------------------------
 100        110 110        116 116        117 117        120 120        124

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 2

Deallocating memory:
Enter process ID: P1

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 4

Merged holes!

MENU:
1. Allocate
```

Name: Krithika Swaminathan
Roll No.: 205001057

```
MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3

Displaying memory:

Allocated memory:

NULL

Free memory:

---------------
|       H       |
---------------
 100         124

Physical memory:

---------------
|       H       |
---------------
 100         124

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3

Displaying memory:

Allocated memory:

NULL

Free memory:

---------------
|       H       |
---------------
 100         124

Physical memory:

---------------
|       H       |
---------------
 100         124

MENU:
1. Allocate
```

Name: Krithika Swaminathan
Roll No.: 205001057

```
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 5

Returning to main program...

Memory allocation:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit from program
Enter choice: 3

__Worst Fit Memory Allocation Algorithm__

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 1

Allocating memory:
Enter process ID: P1
Enter process size: 2
```

```
MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3

Displaying memory:

Allocated memory:

---------------
|      P1     |
---------------
 100        102

Free memory:

------------------------------------------------------------
|      H      ||      H      ||      H      ||      H      |
------------------------------------------------------------
 102        110 110        117 117        120 120        124

Physical memory:

-------------------------------------------------------------------------------
|      P1     ||      H      ||      H      ||      H      ||      H      |
-------------------------------------------------------------------------------
 100        102 102        110 110        117 117        120 120        124
```

Name: Krithika Swaminathan
Roll No.: 205001057

```
MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 4

Merged holes!

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 2

Deallocating memory:
Enter process ID: P1

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3
```

```
Displaying memory:

Allocated memory:

NULL

Free memory:

--------------------------------
|       H       ||      H       |
--------------------------------
 100         102 102         124

Physical memory:

--------------------------------
|       H       ||      H       |
--------------------------------
 100         102 102         124

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 5

Returning to main program...
```

```
Memory allocation:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit from program
Enter choice: 4

Exiting the program...
~/OSL$
```

Name: Krithika Swaminathan
Roll No.: 205001057

Learning outcomes:

- The three memory allocation techniques - first fit, best fit and worst fit - were understood and implemented.
- Methods for allocating and deallocating processes were implemented.
- Availability of free space was increased by coalescing holes in partitions in the memory.
