과목명	운영체제	분반	X	담당교수	김화성 교수님
학과	전자통신공학과	학번	2016707079	이름	하상천
과제명: H/W 10 - deadlock avoidance					

1. 과제설명 (사용자 요구사항 기술: 과제에 대한 설명 및 목표)

강의 교재에서 다음 알고리즘을 구현하고 시험 결과를 제시하라.

1. (Deadlock)의 Banker's Algorithm (37-40)

2. (Deadlock)의 Detection Algorithm (48-49) ?

알고리즘 적용 대상 데이타는 스스로 만들어 사용하나, 객관성있는 데이타가 사용되어야 함.

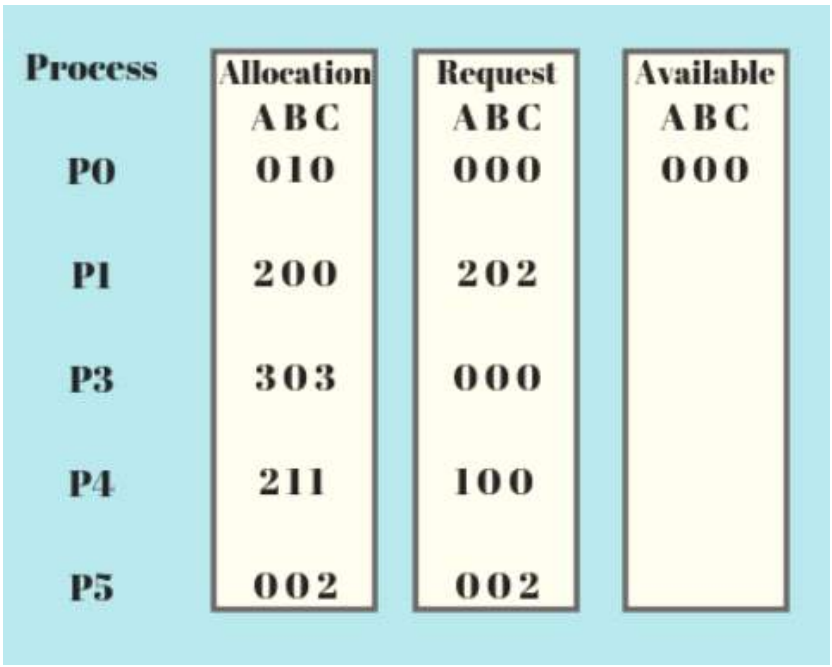
2. 알고리즘 및 자료구조 설계 내용

Let n = number of processes, and m = number of resources types.

- **Available**: Vector of length m. If available [j] = k, there are k instances of resource type R_j available
- **Max**: n x m matrix. If $Max[i,j] = k$, then process P_i may request at most k instances of resource type R_j
- **Allocation**: n x m matrix. If Allocation[i,j] = k then P_i is currently allocated k instances of R_j
- **Need**: n x m matrix. If $Need[i,j] = k$, then P_i may need k more instances of R_j to complete its task

$$Need\ [i,j] = Max[i,j] - Allocation\ [i,j]$$

Banker's Algorithm



Process	Allocation	Request	Available
	A B C	A B C	A B C
P0	0 1 0	0 0 0	0 0 0
P1	2 0 0	2 0 2	
P3	3 0 3	0 0 0	
P4	2 1 1	1 0 0	
P5	0 0 2	0 0 2	

Detection Algorithm

3. 소스코드 설명 (직접 작성한 소스코드중에 핵심 부분을 발췌하여 설명)

```
n = 5; // Number of processes
m = 3; // Number of resources
int alloc[5][3] = { { 0, 1, 0 }, // P0     // Allocation Matrix
{ 2, 0, 0 }, // P1
{ 3, 0, 2 }, // P2
{ 2, 1, 1 }, // P3
{ 3, 3, 2 } }; // P4

int max[5][3] = { { 7, 5, 3 }, // P0     // MAX Matrix
{ 3, 2, 2 }, // P1
{ 9, 0, 2 }, // P2
{ 2, 2, 2 }, // P3
{ 4, 3, 3 } }; // P4
int original_avail[3] = { 0, 0, 2 };    // Available Resources
```

강의 자료에 나와 있는 Banker's algorithm의 예시를 사용했다. P1이 (1,0,2)를 요구했을 때, P4가 (3,3,0)을 요구했을 때, P0가 (0,2,0)을 요구했을 때의 경우들을 구현해보았다.

```
for (i = 0; i < n; i++) {
for (j = 0; j < m; j++)
need[i][j] = max[i][j] - alloc[i][j];
}
int y = 0;
int flag2 = FALSE;
for (k = 0; k < 5; k++) {
for (i = 0; i < n; i++) {
if (f[i] == FALSE) {
int flag = FALSE;
for (j = 0; j < m; j++) {
if (need[i][j] > avail[j]) {
flag = TRUE;
break;
}
}
if (flag == TRUE && i == n - 1) {
flag2 = TRUE;
break;
}

if (flag == FALSE) {
ans[idx++] = i;
for (y = 0; y < m; y++)
avail[y] += alloc[i][y];
f[i] = TRUE;
}
}
}
if (flag2 == TRUE) {
break;
}
}
```

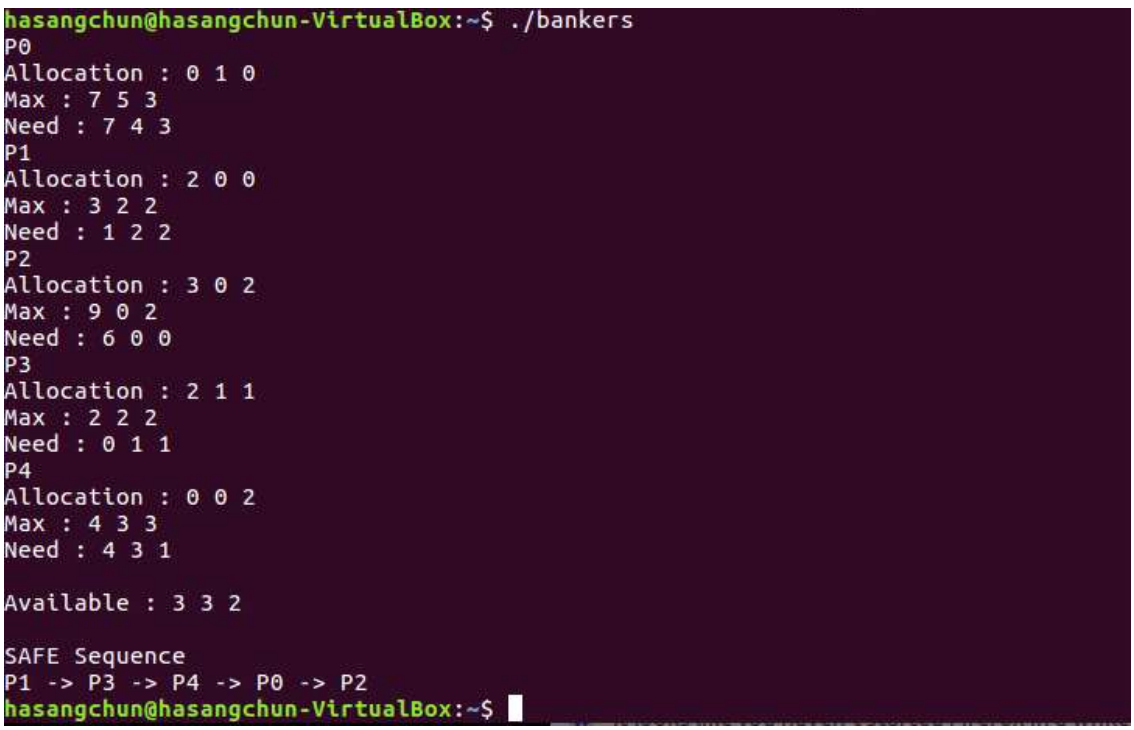
Banker's algorithm에서 max - allocation으로 need를 구하고, need와 avail을 비교하면서 need가 avail보다 작다면 flag를 FALSE로 바꾸어주고, if문을 통해 safe sequence 배열인 ans배열에 프로세스 번호를 넣어주고, allocation을 avail로 더해주어 자원을 반납하였다. need가 avail보다 커서 flag가 계속 TRUE이고, 인덱스도 마지막인 n-1이라면 flag2를 TRUE로 바꾸어주고 break문을 통해 반복문을 탈출한 후 SAFE Sequence가 존재하지 않는다고 출력해주었다. TRUE는 1, FALSE는 0으로 define을 통해 치환해주었다.

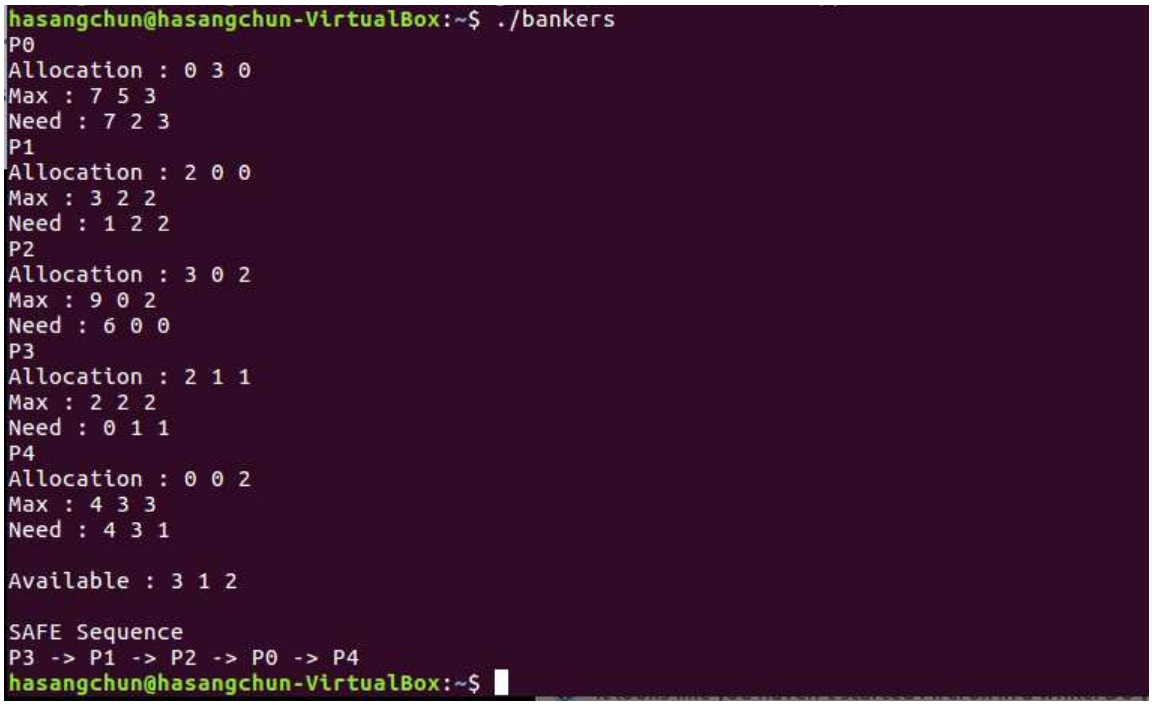
```
for (i = 0; i < n; i++) {
printf("P%d₩n", i);
printf("Allocation : %d %d %d₩n", alloc[i][0], alloc[i][1], alloc[i][2]);
printf("Request : %d %d %d₩n", req[i][0], req[i][1], req[i][2]);
}
printf("₩nAvailable : %d %d %d₩n₩n", original_avail[0], original_avail[1], original_avail[2]);

if (flag2 == TRUE) {
printf("It is deadlocked state.₩n");
}
else {
printf("SAFE Sequence₩n");
for (i = 0; i < n - 1; i++)
printf("P%d -> ", ans[i]);
printf("P%d₩n", ans[n - 1]);
printf("It is not deadlocked state.₩n");
}
```

detection algorithm에서는 allocation, request, available을 사용하였다. safe sequence를 구할 수 있으면 deadlocked state가 아니라고 출력하였고, 그렇지 않은 경우는 deadlocked state라고 출력하였다.

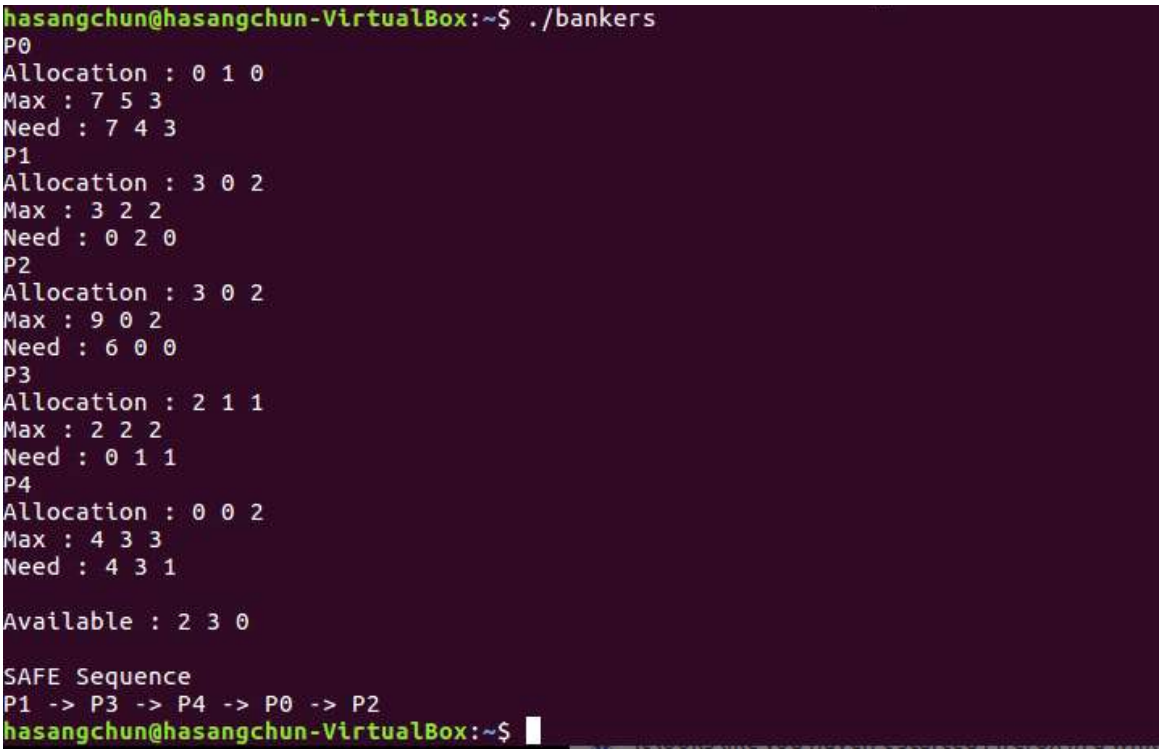
4. 실행결과 및 설명 (실행 결과를 캡쳐하여 첨부한 후 설명)
(그림을 문서에 포함, 글자처럼 취급 옵션, 잉크 절약과 잘 보이게 하기위해 그림 반전)

<Banker's Algorithm>





P0가 (0,2,0)을 요구했을 때



P1이 (1,0,2)를 요구했을 때

```
hasangchun@hasangchun-VirtualBox:~$ ./bankers
P0
Allocation : 0 1 0
Max : 7 5 3
Need : 7 4 3
P1
Allocation : 2 0 0
Max : 3 2 2
Need : 1 2 2
P2
Allocation : 3 0 2
Max : 9 0 2
Need : 6 0 0
P3
Allocation : 2 1 1
Max : 2 2 2
Need : 0 1 1
P4
Allocation : 3 3 2
Max : 4 3 3
Need : 1 0 1

Available : 0 0 2

It doesn't exist SAFE Sequence.
hasangchun@hasangchun-VirtualBox:~$
```

P4가 (3,3,0)을 요구했을 때

<Detection Algorithm>

```
hasangchun@hasangchun-VirtualBox:~$ ./detection
P0
Allocation : 0 1 0
Request : 0 0 0
P1
Allocation : 2 0 0
Request : 2 0 2
P2
Allocation : 3 0 3
Request : 0 0 0
P3
Allocation : 2 1 1
Request : 1 0 0
P4
Allocation : 0 0 2
Request : 0 0 2

Available : 0 0 0

SAFE Sequence
P0 -> P2 -> P3 -> P4 -> P1
It is not deadlocked state.
hasangchun@hasangchun-VirtualBox:~$
```

P2가 (0,0,1)을 요구할 때

5. 고찰 (과제를 진행하면서 배운점 이나, 시행 착오 내용, 기타 느낀점)

이번 과제는 deadlock의 banker's algorithm을 구현하는 것과 deadlock의 detection algorithm을 구현하는 것이었다. banker's algorithm에서는 allocation, max, available, need 배열을 이용해서 구현하였다. max - allocation을 통해서 need배열을 구하고, 반복문과 조건문을 통해 need와 available을 비교하였다. need가 avail보다 큰 경우 break문을 통해 반복문을 탈출하였고, need가 avail보다 작은 경우 allocation을 available에 더해주어 자원을 반납하였고, safe sequence 배열인 ans배열에 프로세스 번호를 넣어주었다. 마지막에는 safe sequence를 출력해주었고, 모든 프로세스의 need가 avail보다 큰 경우는 flag2를 TRUE로 바꾸어주고 break를 통해 반복문을 탈출한 후 safe sequence가 존재하지 않는다고 출력해주었다. Detection algorithm은 allocation, request, available 배열을 이용해서 구현하였는데 위와 마찬가지로 safe sequence가 존재하면 deadlocked state가 아니고, 존재하지 않으면 deadlocked state라고 출력해주었다. Banker's algorithm 같은 경우는 리소스의 최대 개수를 미리 알고 있어야 하기 때문에 실제 돌아가는 프로그램의 적용하기는 어려울 것 같다. 또한 항상 unsafe state를 방지해야 하기 때문에 리소스 활용도가 낮을 것 같다.

6. 전체 소스코드 (글자크기 9에 줄간격을 120%로 유지하고 한 줄이 너무 길지 않게 작성)

<Banker's Algorithm>

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0
int main()
{
// P0, P1, P2, P3, P4 are the Process names

int n, m, i, j, k;
n = 5; // Number of processes
m = 3; // Number of resources
```

```
int alloc[5][3] = { { 0, 1, 0 }, // P0     // Allocation Matrix
{ 2, 0, 0 }, // P1
{ 3, 0, 2 }, // P2
{ 2, 1, 1 }, // P3
{ 3, 3, 2 } }; // P4

int max[5][3] = { { 7, 5, 3 }, // P0     // MAX Matrix
{ 3, 2, 2 }, // P1
{ 9, 0, 2 }, // P2
{ 2, 2, 2 }, // P3
{ 4, 3, 3 } }; // P4
int original_avail[3] = { 0, 0, 2 };    // Available Resources
int avail[3] = { 0, 0, 2 };
int f[n], ans[n], idx = 0;
for (k = 0; k < n; k++) {
f[k] = FALSE;
}
int need[n][m];
for (i = 0; i < n; i++) {
for (j = 0; j < m; j++)
need[i][j] = max[i][j] - alloc[i][j];
}
int y = 0;
int flag2 = FALSE;
for (k = 0; k < 5; k++) {
for (i = 0; i < n; i++) {
if (f[i] == FALSE) {
int flag = FALSE;
for (j = 0; j < m; j++) {
if (need[i][j] > avail[j]) {
flag = TRUE;
break;
}
}
if (flag == TRUE && i == n - 1) {
flag2 = TRUE;
break;
}

if (flag == FALSE) {
ans[idx++] = i;
for (y = 0; y < m; y++)
avail[y] += alloc[i][y];
f[i] = TRUE;
}
```

```
}
}
if (flag2 == TRUE) {
break;
}
}

for (i = 0; i < n; i++) {
printf("P%d₩n", i);
printf("Allocation : %d %d %d₩n", alloc[i][0], alloc[i][1], alloc[i][2]);
printf("Max : %d %d %d₩n", max[i][0], max[i][1], max[i][2]);
printf("Need : %d %d %d₩n", need[i][0], need[i][1], need[i][2]);
}
printf("₩nAvailable : %d %d %d₩n₩n", original_avail[0], original_avail[1], original_avail[2]);

if (flag2 == TRUE) {
printf("It doesn't exist SAFE Sequence.₩n");
}
else {
printf("SAFE Sequence₩n");
for (i = 0; i < n - 1; i++)
printf("P%d -> ", ans[i]);
printf("P%d₩n", ans[n - 1]);
}
return (0);
}
```

<Detection Algorithm>

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0
int main()
{
// P0, P1, P2, P3, P4 are the Process names

int n, m, i, j, k;
n = 5; // Number of processes
m = 3; // Number of resources
int alloc[5][3] = { { 0, 1, 0 }, // P0     // Allocation Matrix
{ 2, 0, 0 }, // P1
{ 3, 0, 3 }, // P2
{ 2, 1, 1 }, // P3
```

```
{ 0, 0, 2 } }; // P4

int req[5][3] = { { 0, 0, 0 }, // P0     // Request Matrix
{ 2, 0, 2 }, // P1
{ 0, 0, 0 }, // P2
{ 1, 0, 0 }, // P3
{ 0, 0, 2 } }; // P4

int avail[3] = { 0, 0, 0 }; // Available Resources
int original_avail[3] = { 0, 0, 0 };
int f[n], ans[n], idx = 0;
for (k = 0; k < n; k++) {
f[k] = FALSE;
}

int y = 0;
int flag2 = FALSE;
for (k = 0; k < 5; k++) {
for (i = 0; i < n; i++) {
if (f[i] == FALSE) {
int flag = FALSE;
for (j = 0; j < m; j++) {
if (req[i][j] > avail[j]) {
flag = TRUE;
break;
}
}
if (flag == TRUE && i == n - 1) {
flag2 = TRUE;
break;
}

if (flag == FALSE) {
ans[idx++] = i;
for (y = 0; y < m; y++)
avail[y] += alloc[i][y];
f[i] = TRUE;
}
}
}
if (flag2 == TRUE) {
break;
}
}
```

```
for (i = 0; i < n; i++) {
printf("P%d₩n", i);
printf("Allocation : %d %d %d₩n", alloc[i][0], alloc[i][1], alloc[i][2]);
printf("Request : %d %d %d₩n", req[i][0], req[i][1], req[i][2]);
}
printf("₩nAvailable : %d %d %d₩n₩n", original_avail[0], original_avail[1], original_avail[2]);

if (flag2 == TRUE) {
printf("It is deadlocked state.₩n");
}
else {
printf("SAFE Sequence₩n");
for (i = 0; i < n - 1; i++)
printf("P%d -> ", ans[i]);
printf("P%d₩n", ans[n - 1]);
printf("It is not deadlocked state.₩n");
}
return (0);

}
```

(글자크기는 10으로 유지하고 줄간격도 160%를 유지할 것)