**BANKERS ALGORITHM**

#include <stdio.h>

#include <stdbool.h>

#define N 5 // Number of processes

#define M 3 // Number of resources

bool is\_safe(int processes[], int avail[], int max\_demand[N][M], int allocation[N][M], int need[N][M], int safe\_sequence[]) {

int work[M];

bool finish[N] = {false};

for (int i = 0; i < M; i++) work[i] = avail[i];

int count = 0;

while (count < N) {

bool found = false;

for (int p = 0; p < N; p++) {

if (!finish[p]) {

bool can\_allocate = true;

for (int j = 0; j < M; j++) {

if (need[p][j] > work[j]) {

can\_allocate = false;

break;

}

}

if (can\_allocate) {

for (int j = 0; j < M; j++) work[j] += allocation[p][j];

safe\_sequence[count++] = p;

finish[p] = true;

found = true;

}

}

}

if (!found) {

return false;

}

}

return true;

}

bool request\_resources(int process\_id, int request[], int processes[], int avail[], int max\_demand[N][M], int allocation[N][M], int need[N][M], int safe\_sequence[]) {

for (int i = 0; i < M; i++) {

if (request[i] > need[process\_id][i]) return false;

if (request[i] > avail[i]) return false;

}

for (int i = 0; i < M; i++) {

avail[i] -= request[i];

allocation[process\_id][i] += request[i];

need[process\_id][i] -= request[i];

}

if (!is\_safe(processes, avail, max\_demand, allocation, need, safe\_sequence)) {

for (int i = 0; i < M; i++) {

avail[i] += request[i];

allocation[process\_id][i] -= request[i];

need[process\_id][i] += request[i];

}

return false;

}

return true;

}

int main() {

int processes[N] = {0, 1, 2, 3, 4};

int avail[M] = {3, 3, 2};

int max\_demand[N][M] = {

{7, 5, 3},

{3, 2, 2},

{9, 0, 2},

{2, 2, 2},

{4, 3, 3}

};

int allocation[N][M] = {

{0, 1, 0},

{2, 0, 0},

{3, 0, 2},

{2, 1, 1},

{0, 0, 2}

};

int need[N][M] = {

{7, 4, 3},

{1, 2, 2},

{6, 0, 0},

{0, 1, 1},

{4, 3, 1}

};

int safe\_sequence[N];

if (is\_safe(processes, avail, max\_demand, allocation, need, safe\_sequence)) {

printf("System is in a safe state.\nSafe sequence: ");

for (int i = 0; i < N; i++) {

printf("%d ", safe\_sequence[i]);

}

printf("\n");

} else {

printf("System is not in a safe state.\n");

}

int request[M] = {1, 0, 2};

int process\_id = 1;

if (request\_resources(process\_id, request, processes, avail, max\_demand, allocation, need, safe\_sequence)) {

printf("Request can be granted.\nSafe sequence: ");

for (int i = 0; i < N; i++) {

printf("%d ", safe\_sequence[i]);

}

printf("\n");

} else {

printf("Request cannot be granted.\n");

}

return 0;

}

Orphan

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main() {

pid\_t pid = fork();

if (pid > 0) {

printf("Parent process (PID: %d) exiting...\n", getpid());

exit(0);

} else if (pid == 0) {

printf("Child process (PID: %d) running...\n", getpid());

sleep(5);

printf("Child process (PID: %d) becoming orphan...\n", getpid());

sleep(10);

printf("Child process (PID: %d) exiting...\n", getpid());

exit(0);

} else {

perror("fork failed");

return 1;

}

}

Zombie

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main() {

pid\_t pid = fork();

if (pid > 0) {

printf("Parent process (PID: %d) sleeping...\n", getpid());

sleep(10);

printf("Parent process (PID: %d) exiting...\n", getpid());

exit(0);

} else if (pid == 0) {

printf("Child process (PID: %d) exiting...\n", getpid());

exit(0);

} else {

perror("fork failed");

return 1;

}

}