Jay Patel

CS 352

HW5

Professor Johnny Wang

7.15:

Due to the cost of keeping track of the current resources allocation, a deadlock avoidance scheme tends to increase the runtime overheads. But, a deadlock-avoidance scheme allows for more concurrent use of resources than schemes that prevents the formation of deadlock. In respect to that, a deadlock avoidance scheme could increase system.

7.16:

a. Increase ***Available*** (new resources added).

Answer: Safe any time and can be changed without any problems.

b. Decrease ***Available*** (resource permanently removed from system).

Answer: Safe only when Max ≤ available and can cause an increase in possibility of deadlock.

c. Increase ***Max*** for one process (the process needs or wants more resources than allowed).

Answer: Safe only when Max ≤ available and can cause an increase in possibility of deadlock.

d. Decrease ***Max*** for one process (the process decides it does not need that many resources).

Answer: Safe any time and can be changed without any problems.

7.18:

Using section 7.6.2 from textbook we know that we have:

- $\sum_{i=1}^{n} Max_i < m + n$
- $Max_i \geq 1$ for all i

$Proof: require_i = max_i - allocation_i$
Let's say if there is a deadlock then:

- $\sum_{i=1}^{n} Allocation_i = m$

use the first equation: $\sum require_i + \sum Allocation_i = \sum Max_i < m + n$
use the third equation: $\sum require_i + m + m < n$
After rewrite: $\sum_{i=1}^{n} require_i < n$
This says that there exists a process P_i such that $require_i = 0$. Since $Max_i \geq 1$ it follows that p_i
has at least one resource that it can release. Hence the system cannot be in deadlock state. There is a contradiction.

7.22:

a) Processes P_2, P_1 and P_3 are able to finish but others can't. So, it's <u>not safe.</u>

b) Processes P_1, P_2, P_3 are able to finish but P_0 and P_4 are also able to finish. So, it's <u>safe.</u>

7.24:

The deadlock detection algorithm assumes that no further resource requests will come in the form the process to complete its task beyond what it currently has. If the process requests more resources later, this assumption is violated and the system could enter a deadlocked state at a later point in time.

6:

```c
/* Include Files */

#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

/* External References */
extern void* world(  );
extern void* hello(  );
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;

void main( int argc, char *argv[] )
{
    pthread_t threadA;
    pthread_t threadB;

    pthread_create(&threadA,NULL, &hello, NULL);
    pthread_create(&threadB,NULL, &world, NULL);

    pthread_join(threadA, NULL);
    pthread_join(threadB, NULL);
    printf("\n");

}
/* world - print the "world" part. */
void *world(  )
{
    pthread_mutex_lock(&m);
     printf( "world" );
    pthread_mutex_unlock(&m);
}
/* hello - print the "hello" part. */
void hello( void )
{
     pthread_mutex_lock(&m);
     printf( "hello " );
    pthread_mutex_unlock(&m);
}
```