On *fork():* What are all possible outputs of the following program? Why?

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   int pid ;

   pid = fork() ;    /* fork another process */

   if (pid == 0)  { /* child process*/

    printf("A") ;

    }  else if (pid > 0 )  {  /* parent process */

    printf("B") ;

   }
}
```

On *fork():* What are all possible outputs of the following program? Why?

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   int pid ;

   pid = fork() ;     /* fork another process */

   if (pid == 0)  { /* child process*/

    printf("A") ;

    }  else if (pid > 0 )  {   /* parent process */

        wait(NULL) ;
        printf("B") ;

   }
}
```

On *fork():* What are all possible outputs of the program?

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   int pid ;
   int control ;

   control = 10 ;

   pid = fork() ;    /* fork another process */

   if (pid == 0)  { /* child process*/

    printf("child=%d\n", control) ;

    }  else if (pid > 0 )  {  /* parent process */

     printf("parent=%d",control) ;

   }
}
```

On *fork():* What are the output of the program?

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int pid ;
  int control ;

  control = 10 ;

  pid = fork() ;    /* fork another process */

  control = 20 ;

  if (pid == 0)  { /* child process*/

   printf("child=%d", control) ;

   }  else if (pid > 0 )  {  /* parent process */

    printf("parent=%d",control) ;

  }
}
```

On *fork():* What are the output of the program?

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int pid ;
  int control ;

  control = 10 ;

  pid = fork() ;    /* fork another process */

  if (pid == 0)  { /* child process*/

   control = 20 ;
   printf("child=%d", control) ;

   }  else if (pid > 0 )  {  /* parent process */

    printf("parent=%d",control) ;

  }
}
```

Synchronization Algorithm Analysis

- Is the following simplified Bakery algorithm a correct solution to the critical section problem?

```
/* shared data */
line 1    int number[n] ;    /* initially all 0 */

   /* The code for process Pi (i=0,1,…,n-1)*/
line 2    number[i] = max(number[0], number[1], ..., number [n-1])+1;
line 3    for (j = 0; j < n; j++) {
line 4        while ((number[j] != 0) && (number[j],j) < (number[i],i)) ;
line 5    }

          critical section

line 6    number[i] = 0;
```

Synchronization Algorithm Analysis

- Is the following simplified Bakery algorithm a correct solution to the critical section problem?

```
/* shared data */
line 1   int number[n] ;    /* initially all 0 */

  /*  The code for process Pi (i=0,1,…,n-1) */
line 2   number[i] = 1;
line 3   for (j = 0; j < n; j++) {
line 4      while ((number[j] != 0) &&  pid(j)< pid(i)) ;
line 5   }

         critical section

line 6   number[i] = 0;
```