CS 4410 Problem Solving Session

September 7 and 8

Problem 1 - Fork & Wait

Function `fork` returns `0` to the child process and the child's process identifier to the parent.
Function `wait` returns `-1` if there is an error, e.g., when the executing process has no child.

Which of the following are valid outputs of the program on the right?

A. 2030401
B. 1234000
C. 2300140
D. 2034012
E. 3200410

```
main() {
   if (fork() == 0) {
      if (fork() == 0) {
         printf("3");
      }
      else {
         pid_t pid;
         int status;
         if ((pid = wait(&status)) > 0) {
            printf("4");
         }
      }
  }
  else {
     if (fork() == 0) {
        printf("1");
        exit(0);
     }
     printf("2");
  }
  printf("0");
  return 0;
}
```

Problem 1 - Fork & Wait

Function `fork` returns `0` to the child process and the child's process identifier to the parent.
Function `wait` returns `-1` if there is an error, e.g., when the executing process has no child.

Which of the following are valid outputs of the program on the right?

A. 2030401
B. 1234000
C. 2300140
D. 2034012
E. 3200410

```
main() {
   if (fork() == 0) {
      if (fork() == 0) {
         printf("3");
      }
      else {
         pid_t pid;
         int status;
         if ((pid = wait(&status)) > 0) {
            printf("4");
         }
      }
  }
  else {
     if (fork() == 0) {
        printf("1");
        exit(0);
     }
     printf("2");
  }
  printf("0");
  return 0;
}
```

Problem 2 - Compare & Swap

Many CPU architectures provide a Compare-And-Swap (CAS) instruction with the following semantics.

Notes:

- `addr` is the address of some memory location.
- CAS requires a *single* read and at most a *single* write cycle.

```
ATOMIC bool CAS(int *addr, int oldval, int newval) {
     if (*addr != oldval) {
          return FALSE;
     }
     *addr = newval;
     return TRUE;
}
```

Problem 2 - Compare & Swap

A. Implement a Test-and-Set function using CAS that does (not counting the fetching instructions) a single memory read and at most a single memory write cycle.

```
bool TAS(int *addr) {
     // your code here
}
```

B. Using CAS, implement an atomic increment operation, `INC(&x)` such that x is incremented by 1.

```
void INC(int *addr) {
     // your code here
}
```

```
ATOMIC bool CAS(int *addr, int oldval, int newval) {
     if (*addr != oldval) {
          return FALSE;
     }
     *addr = newval;
     return TRUE;
}
```

Problem 2 - Compare & Swap

A. Implement a Test-and-Set function using CAS that does (not counting the fetching instructions) a single memory read and at most a single memory write cycle.

```
bool TAS(int *addr) {
     return !CAS(addr, 0, 1);
}
```

B. Using CAS, implement an atomic increment operation, `INC(&x)` such that x is incremented by 1.

```
void INC(int *addr) {
     register int oldval = *addr;
     while(!CAS(addr, oldval, oldval + 1))
          oldval = *addr;
}
```

Why the register variable?

If the compiler is given free reign, it's possible that the oldval will actually reside in the RAM instead of in a register, which adds yet another memory write and read, and increases the chances that the CAS will fail (because it would take longer, and in the interim, another thread can change that value).

```
ATOMIC bool CAS(int *addr, int oldval, int newval) {
     if (*addr != oldval) {
          return FALSE;
     }
     *addr = newval;
     return TRUE;
}
```