Relevant Prototypes

```
// filesystem access
int close(int fd); // ignore retval
int dup(int fd);   //
int dup2(int oldfd, int newfd); // ignore retval
int pipe(int fds[]); // ignore retval
int pipe2(int fds[], int flags); // ignore retval, flags typically O_CLOEXEC
#define STDIN_FILENO 0
#define STDOUT_FILENO 1

// exceptional control flow and multiprocessing
pid_t fork();
pid_t waitpid(pid_t pid, int *status, int flags);
int execvp(const char *path, char *argv[]); // ignore retval
int kill(pid_t pid, int signal); // ignore retval
typedef void (*sighandler_t)(int sig);
sighandler_t signal(int signum, sighandler_t handler); // ignore retval
int sigemptyset(sigset_t *set); // ignore retval
int sigaddset(sigset_t *set, int sig); // ignore retval
int sigprocmask(int how, const
sigset_t *set, sigset_t *old); //
ignore retval

#define WIFEXITED(status)   // macro
#define WIFSTOPPED(status)  // macro

class mutex {
public:
   void lock();
   void unlock();
};

class semaphore {
public:
   semaphore(int count = 0);
   void wait();
   void signal();
};

class condition_variable_any {
public:
   void wait(mutex& m);
   template <typename Pred>
     void wait(mutex& m, Pred p);
   void notify_one();
   void notify_all();
};

class ThreadPool {
public:
   ThreadPool(size_t numThreads);
   void schedule(Thunk t);
   void wait();
};

struct subprocess_t {
  pid_t pid;
  int supplyfd;
  int ingestfd;
};
```

```
template <typename T>
class vector {
public:
   size_t size() const;
   void push_back(const T& elem);
   T& operator[](size_t i);
   const T& operator[](size_t i) const;
};

template <typename T>
class list {
public:
   bool empty() const;
   size_t size() const;
   void push_back(const T& elem);
   T& front();
   void pop_front();
};

template <typename U, typename V>
struct pair {
   U first;
   V second;
};

Template <typename Key, typename Value>
class map {
public:
   // iter points to pair<Key, Value>
   size_t size() const;
   iter find(const Key& k);
   Value& operator[](const Key& k);
};


subprocess_t subprocess(char *argv[],
      bool supplyChildInput,
      bool ingestChildOutput);
```