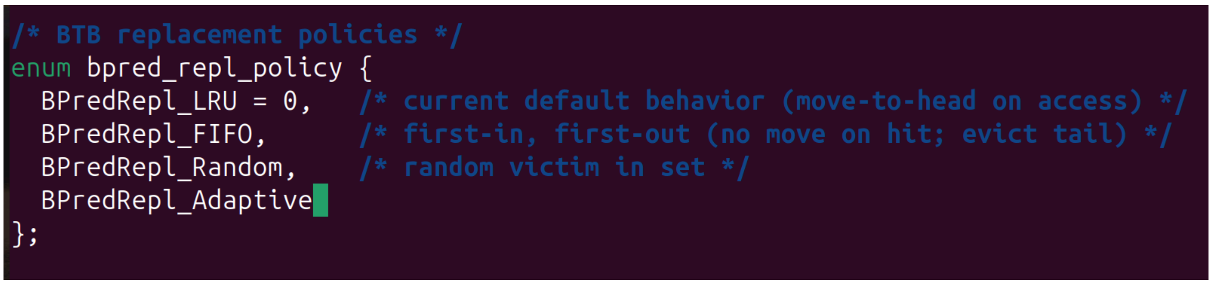
CODES:

These are the main changes that are done to support our project. The codes in which changes have been done are bpred.c , bpred.h , sim-bpred.c , sim-outorder.c(complete codes already provided in git)

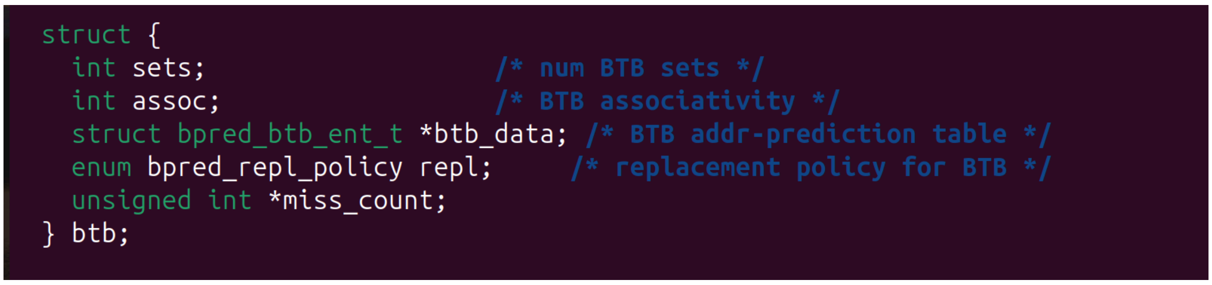
We will be going step by step.

bpred.h:
Insert this block near the top



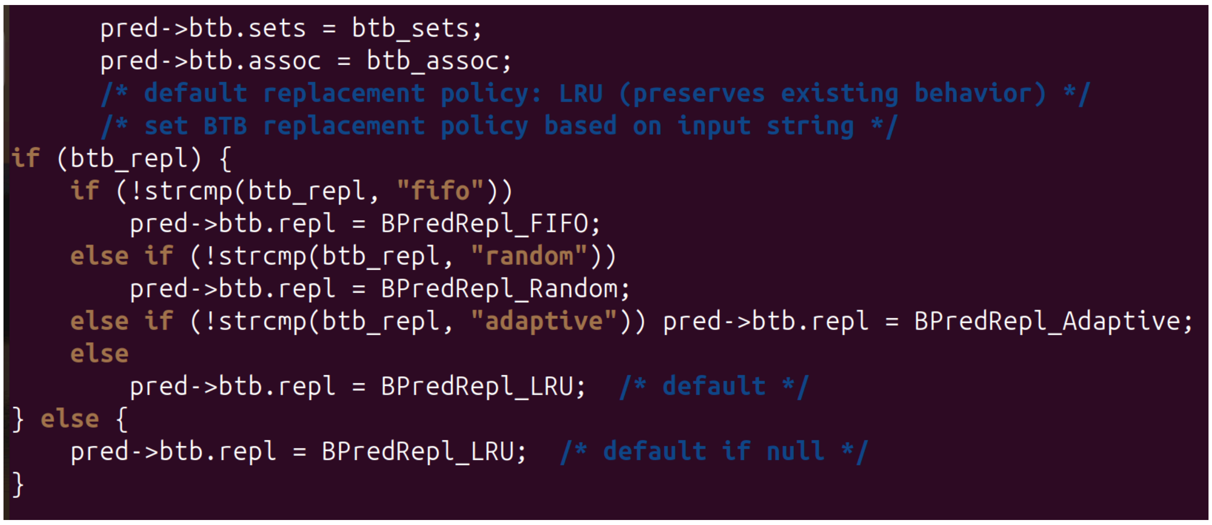
```
/* BTB replacement policies */
enum bpred_repl_policy {
  BPredRepl_LRU = 0,   /* current default behavior (move-to-head on access) */
  BPredRepl_FIFO,      /* first-in, first-out (no move on hit; evict tail) */
  BPredRepl_Random,    /* random victim in set */
  BPredRepl_Adaptive
};
```

In struct bpred_t , find this block and edit as follows:



```
  struct {
    int sets;                   /* num BTB sets */
    int assoc;                  /* BTB associativity */
    struct bpred_btb_ent_t *btb_data; /* BTB addr-prediction table */
    enum bpred_repl_policy repl;     /* replacement policy for BTB */
    unsigned int *miss_count;
  } btb;
```

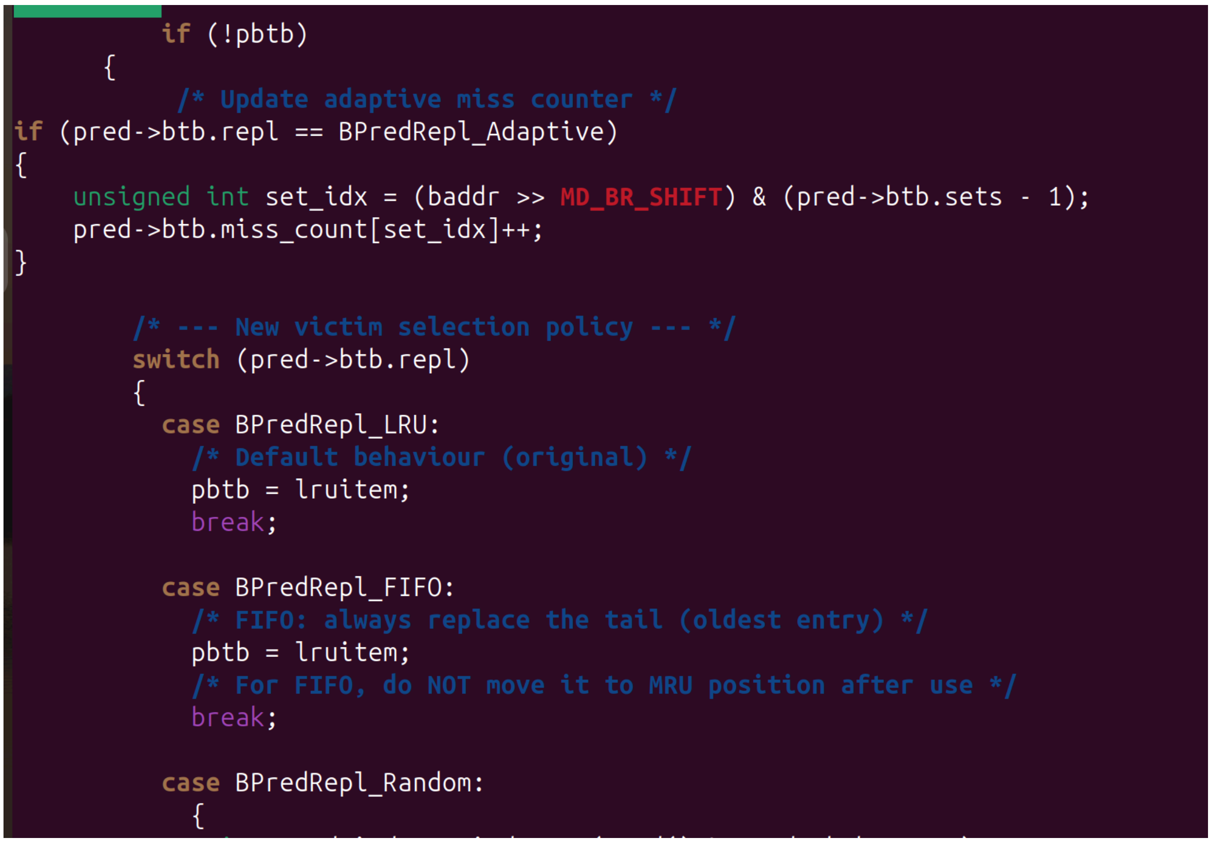
bpred.c:
In bpred_create after the lines pred->btb.sets = btb_sets; pred->btb.assoc = btb_assoc;
Add this block



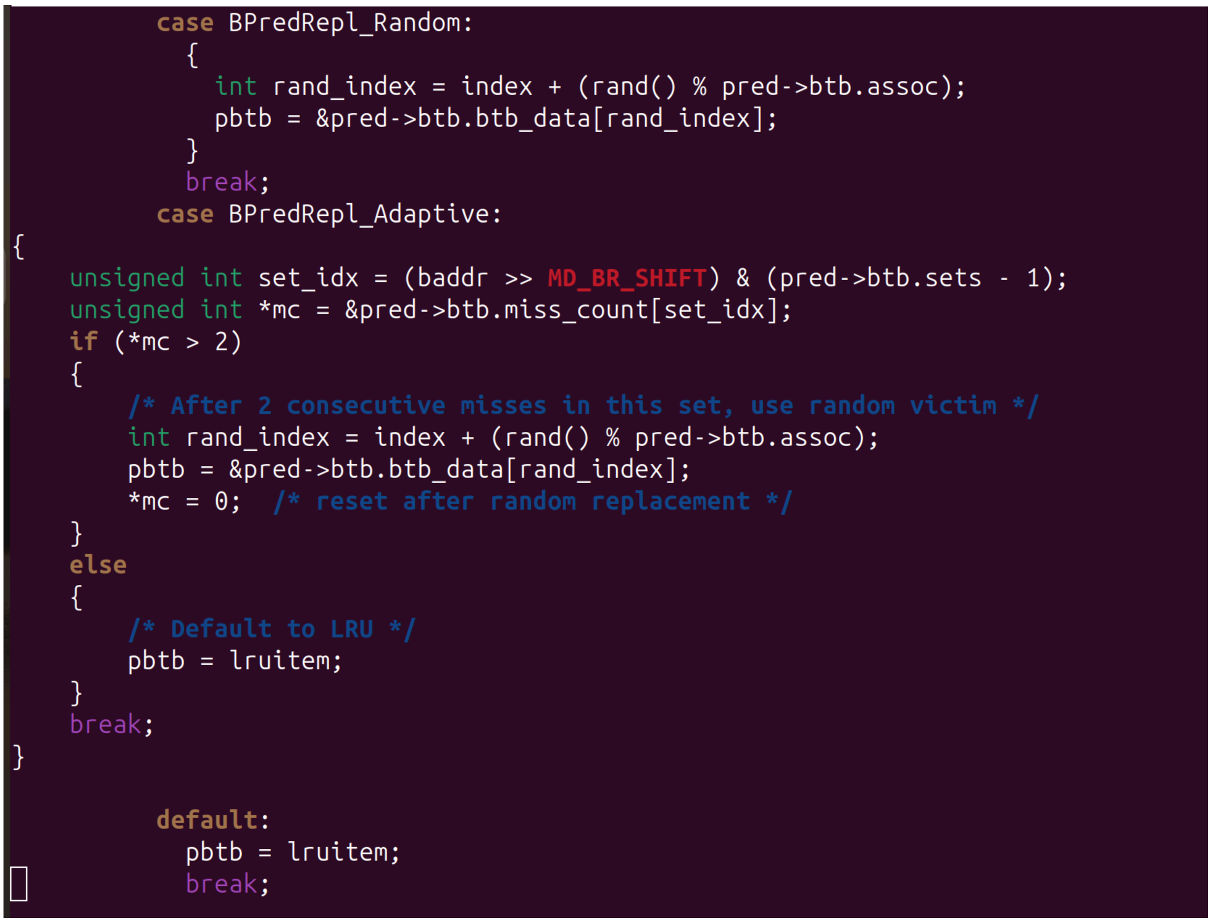
```
      pred->btb.sets = btb_sets;
      pred->btb.assoc = btb_assoc;
      /* default replacement policy: LRU (preserves existing behavior) */
      /* set BTB replacement policy based on input string */
if (btb_repl) {
    if (!strcmp(btb_repl, "fifo"))
        pred->btb.repl = BPredRepl_FIFO;
    else if (!strcmp(btb_repl, "random"))
        pred->btb.repl = BPredRepl_Random;
    else if (!strcmp(btb_repl, "adaptive")) pred->btb.repl = BPredRepl_Adaptive;
    else
        pred->btb.repl = BPredRepl_LRU;  /* default */
} else {
    pred->btb.repl = BPredRepl_LRU;  /* default if null */
}
```

In bpred_update find this line
if (!pbtb)   pbtb = lruitem; change it like this

This is the main replacement policy logic implementation

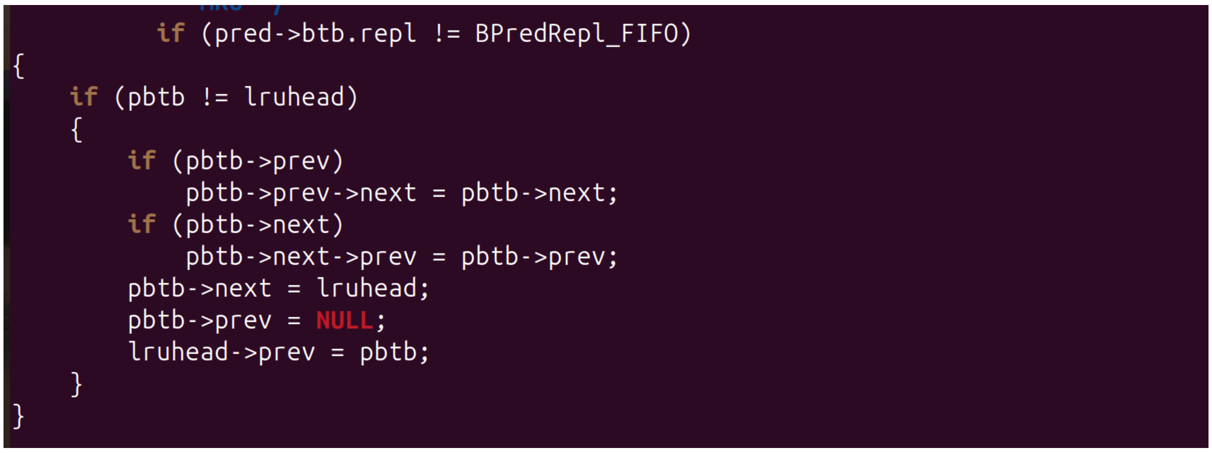


```
          if (!pbtb)
      {
           /* Update adaptive miss counter */
if (pred->btb.repl == BPredRepl_Adaptive)
{
    unsigned int set_idx = (baddr >> MD_BR_SHIFT) & (pred->btb.sets - 1);
    pred->btb.miss_count[set_idx]++;
}

        /* --- New victim selection policy --- */
        switch (pred->btb.repl)
        {
          case BPredRepl_LRU:
            /* Default behaviour (original) */
            pbtb = lruitem;
            break;

          case BPredRepl_FIFO:
            /* FIFO: always replace the tail (oldest entry) */
            pbtb = lruitem;
            /* For FIFO, do NOT move it to MRU position after use */
            break;

          case BPredRepl_Random:
            {
```



```
          case BPredRepl_Random:
            {
              int rand_index = index + (rand() % pred->btb.assoc);
              pbtb = &pred->btb.btb_data[rand_index];
            }
            break;
          case BPredRepl_Adaptive:
{
    unsigned int set_idx = (baddr >> MD_BR_SHIFT) & (pred->btb.sets - 1);
    unsigned int *mc = &pred->btb.miss_count[set_idx];
    if (*mc > 2)
    {
        /* After 2 consecutive misses in this set, use random victim */
        int rand_index = index + (rand() % pred->btb.assoc);
        pbtb = &pred->btb.btb_data[rand_index];
        *mc = 0;  /* reset after random replacement */
    }
    else
    {
        /* Default to LRU */
        pbtb = lruitem;
    }
    break;
}

          default:
            pbtb = lruitem;
            break;
```

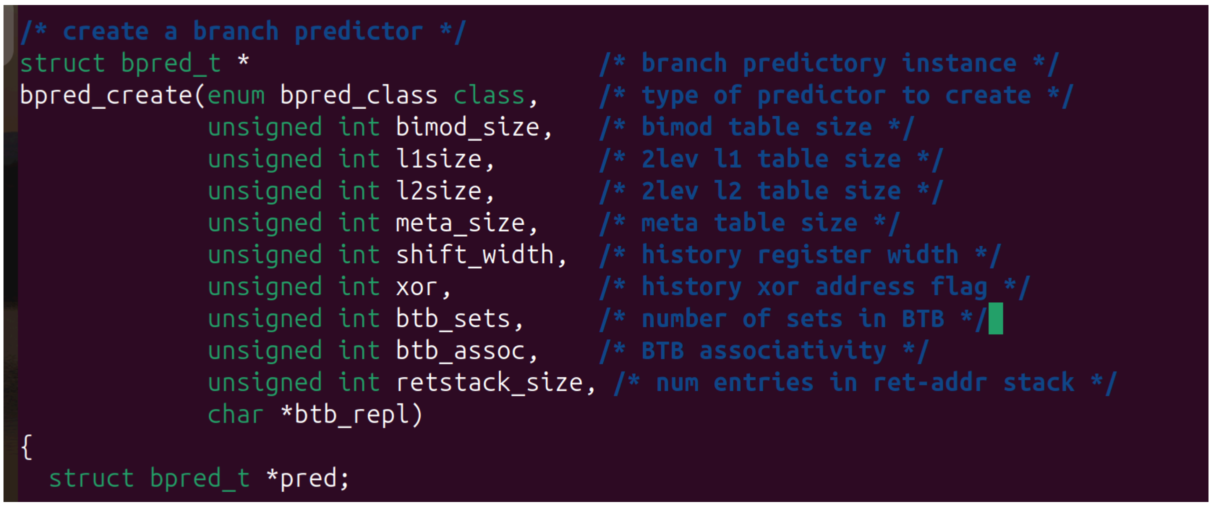
Just a few lines after , we see   if (pbtb != lruhead){ … } . Wrap it up like this
This block differentiates LRU and FIFO.



```
          if (pred->btb.repl != BPredRepl_FIFO)
{
    if (pbtb != lruhead)
    {
        if (pbtb->prev)
            pbtb->prev->next = pbtb->next;
        if (pbtb->next)
            pbtb->next->prev = pbtb->prev;
        pbtb->next = lruhead;
        pbtb->prev = NULL;
        lruhead->prev = pbtb;
    }
}
```

For introducing replacement policy in command line following changes are needed:

bpred.c:
At the top of code, find bpred_create and add this parameter **char btb_repl**

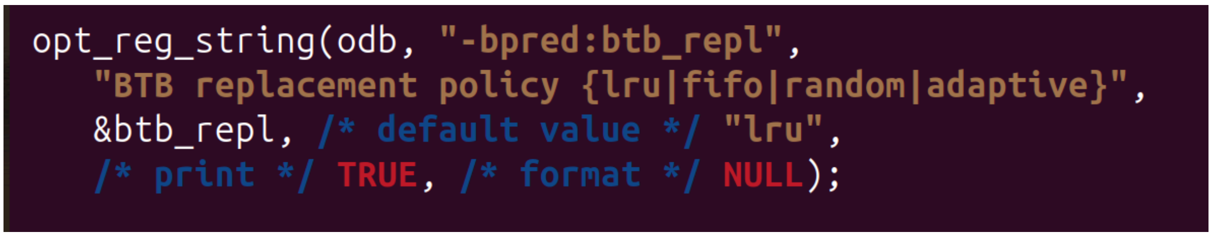


```
/* create a branch predictor */
struct bpred_t *                        /* branch predictory instance */
bpred_create(enum bpred_class class,    /* type of predictor to create */
             unsigned int bimod_size,   /* bimod table size */
             unsigned int l1size,       /* 2lev l1 table size */
             unsigned int l2size,       /* 2lev l2 table size */
             unsigned int meta_size,    /* meta table size */
             unsigned int shift_width,  /* history register width */
             unsigned int xor,          /* history xor address flag */
             unsigned int btb_sets,     /* number of sets in BTB */
             unsigned int btb_assoc,    /* BTB associativity */
             unsigned int retstack_size, /* num entries in ret-addr stack */
             char *btb_repl)
{
  struct bpred_t *pred;
```

sim_bpred.c:

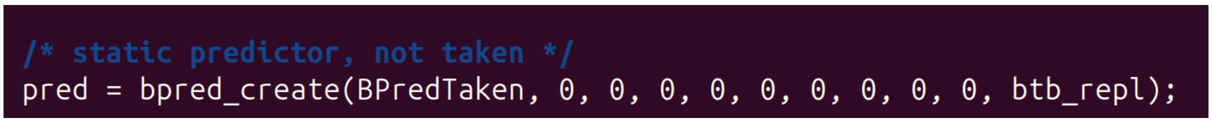
Add this at the top near other declarations
static char *btb_repl = "lru";

In sim_reg_options , add this block

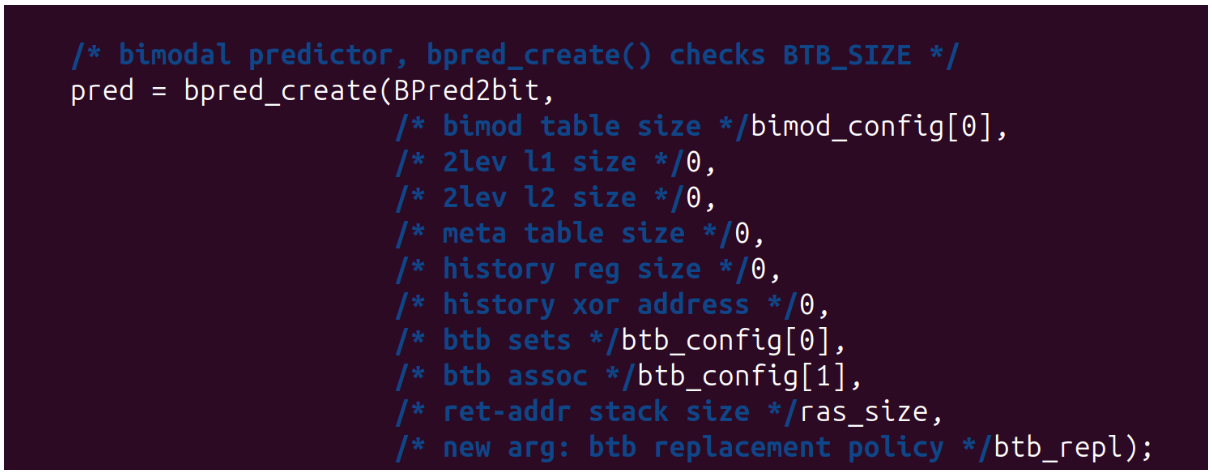


```
opt_reg_string(odb, "-bpred:btb_repl",
    "BTB replacement policy {lru|fifo|random|adaptive}",
    &btb_repl, /* default value */ "lru",
    /* print */ TRUE, /* format */ NULL);
```

Whenever the bpred_create func is called in the code, update it to have the extra parameter like this



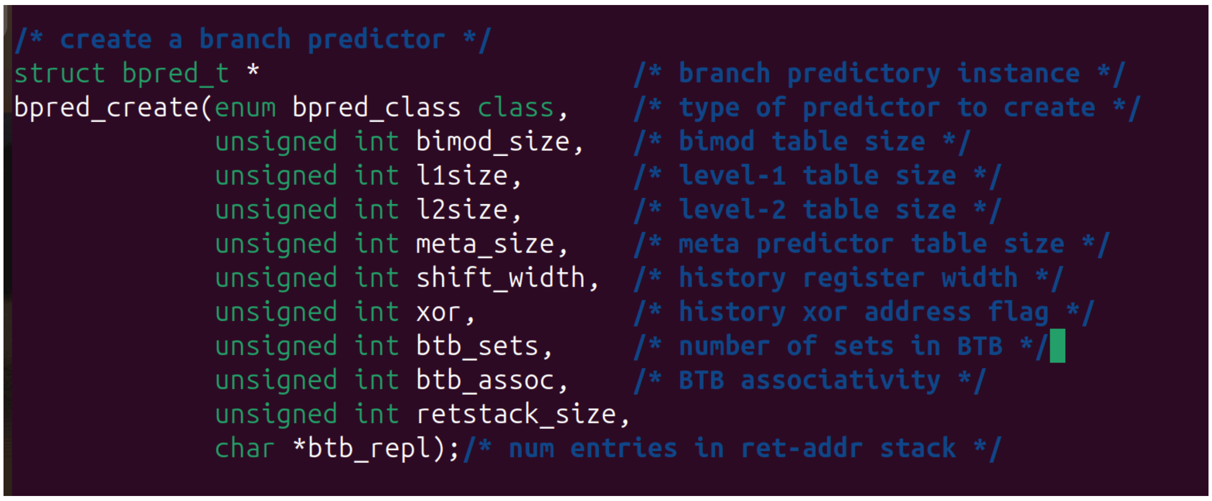
```
/* static predictor, not taken */
pred = bpred_create(BPredTaken, 0, 0, 0, 0, 0, 0, 0, 0, 0, btb_repl);
```



```
    /* bimodal predictor, bpred_create() checks BTB_SIZE */
    pred = bpred_create(BPred2bit,
                        /* bimod table size */bimod_config[0],
                        /* 2lev l1 size */0,
                        /* 2lev l2 size */0,
                        /* meta table size */0,
                        /* history reg size */0,
                        /* history xor address */0,
                        /* btb sets */btb_config[0],
                        /* btb assoc */btb_config[1],
                        /* ret-addr stack size */ras_size,
                        /* new arg: btb replacement policy */btb_repl);
```

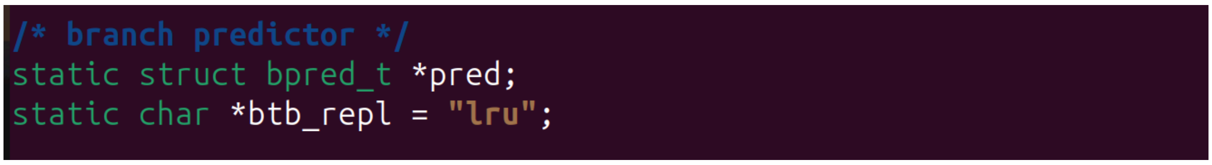
bpred.h:

Update the call here as well

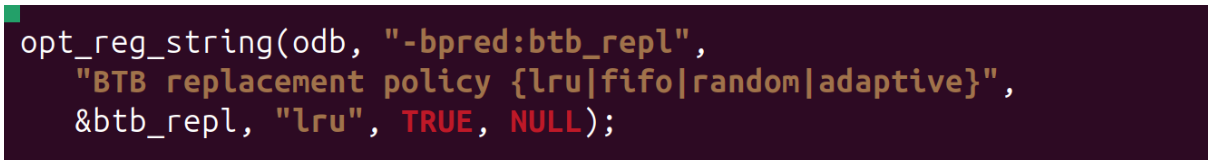


```
/* create a branch predictor */
struct bpred_t *                        /* branch predictory instance */
bpred_create(enum bpred_class class,    /* type of predictor to create */
             unsigned int bimod_size,   /* bimod table size */
             unsigned int l1size,       /* level-1 table size */
             unsigned int l2size,       /* level-2 table size */
             unsigned int meta_size,    /* meta predictor table size */
             unsigned int shift_width,  /* history register width */
             unsigned int xor,          /* history xor address flag */
             unsigned int btb_sets,     /* number of sets in BTB */
             unsigned int btb_assoc,    /* BTB associativity */
             unsigned int retstack_size,
             char *btb_repl);/* num entries in ret-addr stack */
```

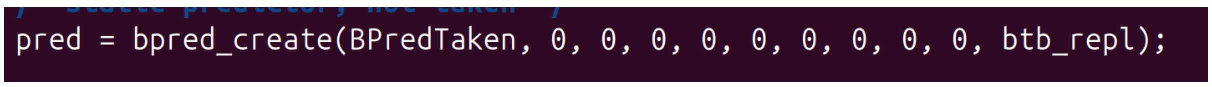
sim-outorder.c:
Just like above make these changes in sim-outorder.c as well:



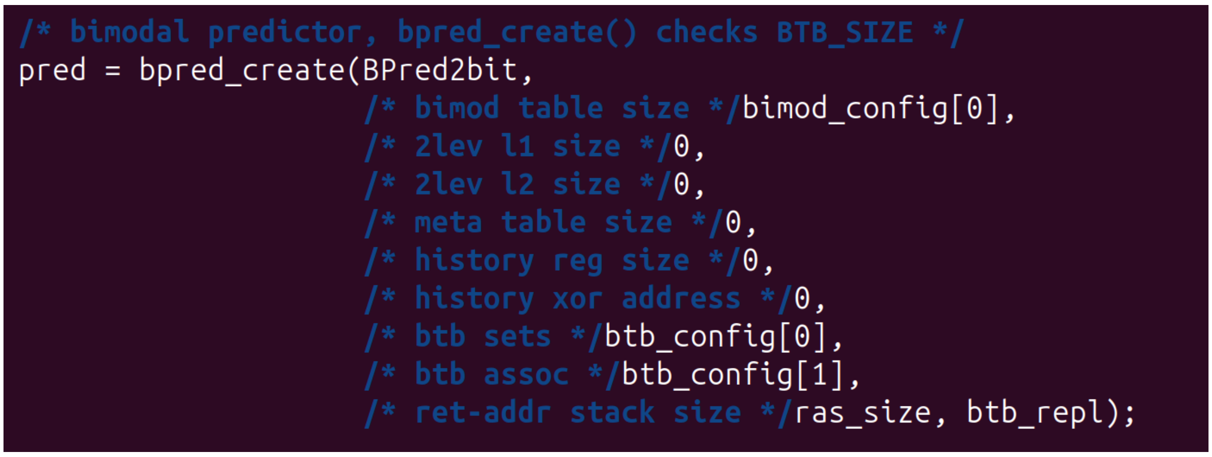
```c
/* branch predictor */
static struct bpred_t *pred;
static char *btb_repl = "lru";
```



```c
opt_reg_string(odb, "-bpred:btb_repl",
   "BTB replacement policy {lru|fifo|random|adaptive}",
   &btb_repl, "lru", TRUE, NULL);
```



```c
pred = bpred_create(BPredTaken, 0, 0, 0, 0, 0, 0, 0, 0, 0, btb_repl);
```



```c
/* bimodal predictor, bpred_create() checks BTB_SIZE */
pred = bpred_create(BPred2bit,
                    /* bimod table size */bimod_config[0],
                    /* 2lev l1 size */0,
                    /* 2lev l2 size */0,
                    /* meta table size */0,
                    /* history reg size */0,
                    /* history xor address */0,
                    /* btb sets */btb_config[0],
                    /* btb assoc */btb_config[1],
                    /* ret-addr stack size */ras_size, btb_repl);
```

(Do for all the function calls)

These are the main changes that have been made , any other small changes to be made can be identified when doing make through errors.

If all necessary changes are done successfully, do

make clean

make

If it is successful, we are ready to run the commands and start gathering results.

The changes for adaptive policy aren't mentioned here, but the full code available already has the changes done.